
# EXPLOITATION OF SELF ORGANIZATION IN UAV SWARMS FOR OPTIMIZATION IN COMBAT ENVIRONMENTS

THESIS

Dustin J. Nowak, Captain, USAF

AFIT/GCS/ENG/08-18

**DEPARTMENT OF THE AIR FORCE**
**AIR UNIVERSITY**

***AIR FORCE INSTITUTE OF TECHNOLOGY***

**Wright-Patterson Air Force Base, Ohio**

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCS/ENG/08-18

# EXPLOITATION OF SELF ORGANIZATION IN UAV SWARMS FOR OPTIMIZATION IN COMBAT ENVIRONMENTS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Dustin J. Nowak, B.S.C.S

Captain, USAF

March 2008

AFIT/GCS/ENG/08-18

# EXPLOITATION OF SELF ORGANIZATION IN UAV SWARMS FOR OPTIMIZATION IN COMBAT ENVIRONMENTS

Dustin J. Nowak, B.S.C.S

Captain, USAF

Approved:

| | |
|---|---|
| /signed/ | 3 Mar 2008 |
| Dr. Gary B. Lamont, PhD (Chairman) | date |
| /signed/ | 3 Mar 2008 |
| Dr. Gilbert Peterson, PhD (Member) | date |
| /signed/ | 3 Mar 2008 |
| Maj. Michael J. Veth, PhD (Member) | date |

# *Abstract*

This investigation focuses primarily on the development of effective target engagement for unmanned aerial vehicle (UAV) swarms using autonomous self-organized cooperative control. This development required the design of a new abstract UAV swarm control model which flows from an abstract Markov structure, a Partially Observable Markov Decision Process. Self-organization features, bio-inspired attack concepts, evolutionary computation (multi-objective genetic algorithms, differential evolution), and feedback from environmental awareness are instantiated within this model. The associated decomposition technique focuses on the iterative deconstruction of the problem domain state and dynamically building-up of self organizational rules as related to the problem domain environment. Resulting emergent behaviors provide the appropriate but different dynamic activity of each UAV agent for statistically accomplishing the required multi-agent temporal attack task. The current application implementing this architecture involves both UAV flight formation behaviors and UAVs attacking targets in hostile environments. This temporal application has been quite successful in computational simulation (animation) with supporting statistical analysis. The effort reflects a considerable increase in effectiveness of UAV attacks related to a previous work with increased damage and decreased causalities. In the process of developing this capability an innovative paradigm shift in autonomous agent system design evolved. Heretofore, large dimensional agent systems were developed with an a priori fixed structure, usually with emphasis on top-down or bottom-up management, control, and sensor communication. Because of the fixed structure, extension to very large dimensional systems is generally impractical. This new autonomous self-organized approach dynamically evolves an entangled communication and cooperative control distributed architecture. This entangled architecture

paradigm can be applied to the research development of various large dimensional agent based autonomous systems, military and industrial.

*Acknowledgements*

First and foremost, I would like to thank God for this opportunity, set of gifts, and support. Secondly, I would like to thank my wife, who stands as my greatest inspirations. Thirdly, I would like to thank my three kids, to whom I owe thanks for the gift of perspective. Next, I would like to thank my mother who had the tedious task of proof reading this document. And Finally, I would like to thank my thesis committee for their direction and patience.

Dustin J. Nowak

*Table of Contents*

Page

# *List of Figures*

# *List of Tables*

# *List of Abbreviations*

| Abbreviation | | Page |
|---|---|---|
| UAV | Unmanned Arial Vehicle | 1 |
| DoD | Department of Defense | 1 |
| SWA | Southwest Asia | 1 |
| OIF | Operation Iraqi Freedom | 2 |
| SEAD | Suppression of Enemy Air Defenses | 3 |
| SO | Self Organized | 3 |
| AFRL | Air Force Research Laboratories | 6 |
| VCL | Virtual Combat Laboratory | 6 |
| IADS | Integrated Air Defenses Systems | 6 |
| ANT | Advanced Navigation Technology | 6 |
| FSA | Fuzzy State Aggregation | 11 |
| PCA | Principal Component Analysis | 12 |
| I-POMDP | Interactive Partially Observable Markov Decision Process | 14 |
| BA | Behavior Archetypes | 19 |
| PDES | Parallel Discrete Event Simulator | 36 |
| ACO | Ant Colony Optimization | 37 |
| PSO | Particle Swarm Optimization | 37 |
| AIS | Artificial Immune Systems | 37 |
| MOP | Multi-Objective Optimization Problem | 38 |
| MOEA | Multi-Objective Evolutionary Algorithm | 38 |
| POMDP | Partially Observable Markov Decision Process | 40 |
| TA | Threat Areas | 41 |
| TTP | Tactics, Techniques and Procedures | 41 |
| AFDD1 | Air Force Doctrine Document 1 | 42 |
| GA | Genetic Algorithm | 50 |

| Abbreviation | | Page |
|---|---|---|
| SOGA | Self Organized Genetic Algorithms | 56 |
| GDC | Genetic Distribution Collection | 58 |
| CAA | Correcting Allele Attraction | 58 |
| MOMGA | Multi-Objective Messy Genetic Algorithms | 60 |
| DE | Differential Evolution | 75 |

# *List of Algorithms*

# EXPLOITATION OF SELF ORGANIZATION IN UAV SWARMS FOR OPTIMIZATION IN COMBAT ENVIRONMENTS

## I. Introduction

As military operations move into the $21^{st}$ century and the civilian population clamors for more efficient, effective ways to engage the enemy, autonomous vehicles are moving into the spotlight. For the U.S. Air Force, Unmanned Aerial Vehicles (UAV) are the new wave of technological golden bullets. Autonomy Theory and UAV swarms come together to take the form of Self Organized Swarms for effective control. The goal is the creation of a set of inexpensive vehicles that can carry out a dynamic set of tasks with low communication overhead and low user interaction. In this study we consider at not only the autonomous swarm but how to effectively employ it in a combat environment.

This chapter sets the tone for the entire thesis. It introduces the constraints facing today's Air Force and the current technology behind UAV Swarm technology. The research goals are defined, assumptions and approaches are over viewed and the sponsorship gives its direction. Finally the structure for this document is outlined.

### *1.1 Problem Overview*

After actions in Kosvo in the late 1990's and early into this millennium the United States Department of Defense (DoD) has been actively engaged in increasing the capabilities of UAVs. Most recently the 2007 DoD budget report from the White House sets the immediate goal to increase the number of UAV orbits in Southwest Asia (SWA) from 12 to 21. [14]

Figure 1.1: The Raven, a hand-launched UAV by Aeroviroment [3]

In 2005 the United States Air Force Strategic Planning directorate published The U.S. Air Force Remotely Piloted Aircraft and Unmanned Aerial Vehicle Strategic Vision [18]. It stated that the Air force must be committed to development:

> "...in the areas of on-board data analysis, auto-target recognition, autonomous flight capabilities, autonomous sensor operation, and data compression." [18]

In a 2004 study, posted by the Secretary of Defense, one of the primary goals is to exploit the ability to reduce the size of UAVs and increase the autonomous capabilities [69]. Also noted in this document, is that over 100 UAVs of more than 10 different types were used in Operation Iraqi Freedom (OIF). These documents bolster the development of UAVs, capable of autonomous, heterogeneous, integrated, reconnaissance and target engagement.

Figure 1.2: The second generation predator with munition carrying hard-points. The precursor to platforms specifically designed as UCAVs.

What is the role of the UAV in the modern day military? Currently there are dozens of systems employed by the US military [18]. Some, like those seen in Figure 1.1 are simply for localized surveillance, border and fence patrol. Others like the Predator shown in Figure 1.2 have longer ranges, more sensors and munitions. In LtCol. Clarke's [17] cadre paper, he defines the roles where UAVs are most likely to be employed: *Dull, Dirty, and Dangerous*. Expanding on the definitions adds clarity for the Air Force's intended use of UAVs.

1. Dull: Includes missions with high loiter times, surveillance for instance.
2. Dirty: Includes missions with possible exposure to an unsafe environment, such as WMD impact zones.
3. Dangerous: Although there is a lot of room for interpretation of the term danger, the intent here is to focus on missions like Suppression of Enemy Air Defenses (SEAD).

Although all three need to be accomplished, the first two are more mundane, only requiring the ability to navigate through fairly static spaces. Rarely do these missions include engagement in dynamic environments. Those characteristics are what make the dangerous missions the center of most research efforts. Due to bandwidth constraints resulting from the President's pledge to increase UAVs orbits in SWA [14], they must accomplish these dangerous mission autonomously.

## *1.2 Proposed Research*

In recent years, one explored answer to the autonomous control question is Self Organization(SO) [22, 59, 74]. Simply stated, SO is the process in which agents with out global knowledge come together to form emergent behaviors that accomplish more than an single agent is capable. Here the application of SO takes the idea of emergent behaviors, shown in biological processes, to solve autonomous control problems. The termite mound, shown in Figure 1.3, illustrates the results of this type of behavior; here each termite has a rule set and those rules sets define the termites

Figure 1.3: The emergent structure of the termite hill based on the set of rules implemented throughout a swarm of thousands of agents.

actions without explicit direction or global knowledge. Appendix A shows a myriad of other Biologically Inspired SO Operators (or Abstract Model Types) that have shown success in computational models. By utilizing the UAVs as a group of bio-inspired agents with a basic rule set, swarming properties and task completion emerge.

*1.2.1 Research Goals & Objectives.* Numerous research efforts into creating a self organizing swarm of Unmanned Ariel Vehicles (UAV) that maneuver through an environment exist [22, 59, 73]. Very few techniques have been investigated that enable a swarm to learn how to engage a target set in a successful manner, once the swarm enters a target area. This is a critical step in creating UAV autonomous swarms that can be successfully employed in war fighting roles. There are two tasks a swarm must complete of this investigation, reaching the target area, shown in Figure 1.4, and exploitation of target weaknesses, shown in Figure 1.5.

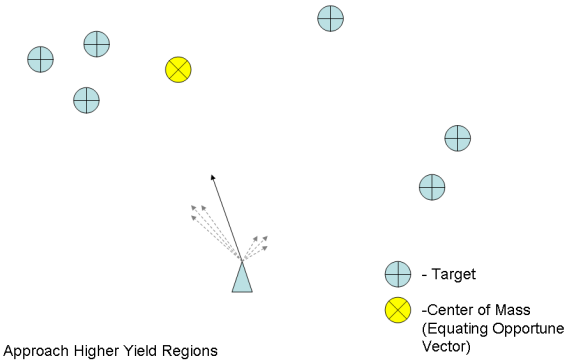


Figure 1.4: Agents find target sets by migrating towards local waypoints. The Triangle represents the UAV and the vectors extending from it represent the distance weighted vector for a given target. The solid line vector represents the vector sum and resulting movement. Through this approach the agent is pulled toward higher yield areas.

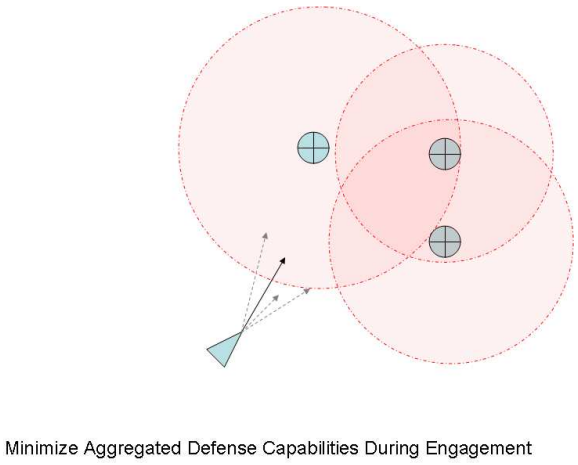


Figure 1.5: Here the agent represented by the triangle again uses a vector sum. The weighting of the target vectors is now defined by the perceived threat. If a target has overlapping engagement rings it is a less attractive target and therefore has a low magnitude vector. The intent is to exploit target set weakness: engaging the middle target causes aggregated defense effects, where engaging the top most target alleviates aggregate capabilities.

In this thesis investigation the goal is the development of a SO model that enables a swarm of UAVs to effectively engage a target sets in the environment. As part of this effort there are three major objectives:

- *Develop* an approach to mathematical modelling through SO problem decomposition resulting in emergent SO structure and as a result extending the SO rules that govern such behavior.
- *Integrate* the resulting behaviors and entangled hierarchy into Swarmfare [59] UAV simulation environment to include these models.
- *Validate* this new model's success through statistical analysis and evaluation of the resulting swarm behavior.

Measured success in the define objective comes from the creation of a thorough problem representation extended from a known problem space. The subjectivity here allows for continual renewal and redesign based on knowledge gained from this work. Measuring rule sets that extend the simulation environment is done by effective extension of the simulation and the library of behaviors in the software. Validation of this new and innovative approach results from the interpolation of the new form of the emergent behaviors of the system in simulation.

### *1.3 Sponsor*

Research into Swarming UAVs interests Air Force Research Laboratories (AFRL) Virtual Combat Laboratory (VCL) AFRL/SNZW. The team lead is Mr. Mike Foster, mike.foster@wpafb.af.mil. The employment int ent focuses on developing low cost UAVs, that can operate in large swarms, numbering into the hundreds. The mission is the jamming and electronic warfare against enemy Integrated Air Defenses Systems (IADS). The VCL's objective is to eliminate IAD threat through electronic or kinetic neutralization.

The AFIT Advanced Navigation Technology (ANT) wishes to further work in this area. In discussions with Maj Michael Veth, member of the ANT team, he

explained that the need for swarm control in small enclosed areas such as caves or buildings exists. Autonomous cooperative control is crucial in this type of environment because communications is severely hampered.

## *1.4 Assumptions, Approach, and Risks*

This section describes the assumptions used to scope the problem, the approach used to solve the problem and the inherent risks in this domain.

*1.4.1 Assumptions.* Assumptions about the UAVs capabilities include:

- Markov independence assumptions
- That sensors capable of detecting adjacent swarm agents exist.
- Fundamental communication links between adjacent swarm agents.
- Sensors to pseudo-accurately detect detection and engagement rings of targets.
- The agents have a knowledge of the complexity of the real-space target area.
- Attacking a single target is not difficult.

*1.4.2 Approach.* Our goal is attainment of truly SO swarm behavior in the combat environment through utilization of the U-Decomposition technique first illustrated in [53]. First the outline of mathematical model that defines the over arching problem domain. Next, decomposition of the target engagement aspect of the problem develops a subset of needed state action pairs. Then derivation of a set of SO rules that address individual sub-problems of swarm movement and target engagement. From the set of SO behaviors a resulting structure emerges through human and Genetic Algorithm[1] (GA) manipulations. The SO rules form emergent swarm behaviors as well. The interaction amongst agents through simple rules and communications

[1]Genetic Algorithms search through a large solution set using a population that is a subset of solutions. Each generation each individual solution is evaluated and the best are mutated and recombined in order to optimize the best of the solutions.

without global knowledge form higher level reactions to the environment This approach address the large problem space. We use a stochastic search technique, a GA, to optimize appropriate weights and controls for the rules set in order to accomplish realtime swarming and target engagement.

*High Level Data Flow*

This section shows the basic flow of the system. In order to accomplish free movement in a space weighted vectors constrained by dynamics models defined the next position. Sets of vectors form different control modes, reconnaissance, attack, loiter, etc. The controller arbitrates between modes. A genetic algorithm "optimizes" the control structure and movement vectors based on parameter weights. Figure 1.6 shows the basic flow that is used throughout the thesis effort.

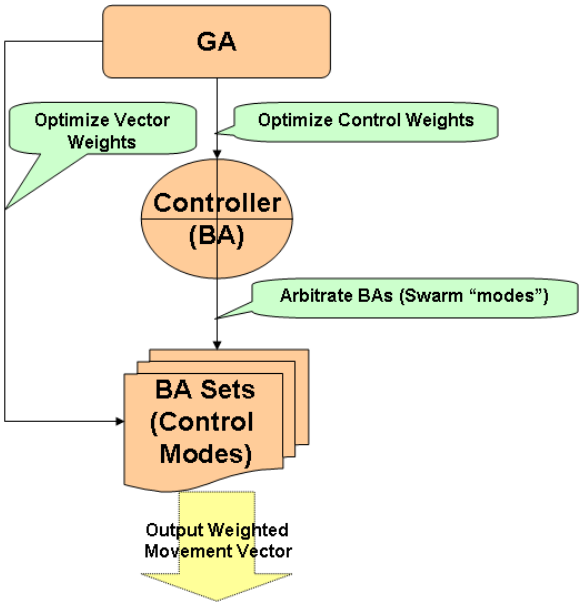


Figure 1.6: The high level data flow showing the GA used to define control weights, the BA controller to arbitrate the BAs, and the BAs pushing a weighted movement vector for control.

*1.4.3 Risks.* In this research effort, the GA techniques are used to develop a weighting system for a set of SO rules that define the behaviors. Genetic algorithms are known to find the local optimal but due to their stochastic nature one can never assure that optimums are found. Also, SO rule sets depend on the emergent behaviors

of the swarm to be effective. These emergent behaviors are sometimes unpredictable and difficult to control. Using both of these techniques the desired results are therefore subject to variations in effectiveness during *development* phases.

## *1.5 Thesis Layout*

In this first Chapter, we have laid out the introduction to this research effort. The second Chapter discusses the background of the problem domain and similar approaches to finding solutions. Because of the size and decomposition of the Problem Domain (high-level) and Algorithm Domain implementation (low-level) designs, they are discussed in two separate chapters, three and four. After that, Chapter Five articulates the design of the experiments to accomplish the validation of the rules and GA techniques. Chapter Six records and reduces the data for analysis. Conclusions, recommendations and suggested future research sections compose the final chapter.

# II. Background

This chapter describes the state of the practice in several areas providing a foundation to this research investigation. Thus, we discuss the background for three prominent aspects of research in UAV swarms: the problem model, autonomous control (including SO), simulation environments, and evolutionary algorithms. The Partially Observable Markov Decision Process (POMDP) is used as a mathematical model for several previous works involving UAV swarm. For this reason it is discussed in detail. Through this construct, we describe the precise dynamic nature of movement through a domain space and the uncertain nature of the domain's response when acted upon by the agents. Many different control structures are used in previous work on autonomous control. Section 2.2, presents some of these control structures. The discussion then narrows to Self Organization approaches. SO provides the governing structure that allows the construction of swarms and their emergent behavior. This results in simple, dynamic, autonomous and fast decision making. A short discussion of existing simulations exists to provide background in the application SO principles in computation. Finally a summary of different Evolutionary Algorithm approaches that can be used for developing autonomous control behaviors are presented.

## *2.1 Partially Observable Markov Decision Process*

Working with UAV swarms in unknown environments is represented by stochastic processes. These processes are commonly modelled by one of the set of Markovian models. Table 2.1 shows the questions defined by Littman [45] to determine the best fit model.

| Markov Models | | Do we have Control over the state transitions? | |
|---|---|---|---|
| | | NO | YES |
| Are the States completely observable? | YES | Markov Chain | Markov Decision Process (MDP) |
| | NO | Hidden Markov Model (HMM) | Partially Observable Markov Decision Process (POMDP) |

Table 2.1: Markov Models [45]

Considerable research into similar problems indicates use of POMDPs as a model. There are three papers that model similar problems with POMDPs: Roy [65], Nikovski [52] and Wardell [80]. Khosla's [41] paper present another way of modeling the paper the provides insight and juxtaposition to the POMDP mapping.

Wardell's thesis [80] mapped movement of autonomous agents to a POMDP. In his domain the agents are playing soccer in a discrete environment. But the agents have no knowledge of the actions of other agents. This uncertain nature of the state and inability to understand how actions effect the environment fits the POMDP defining characteristics. His solution focused on Reinforcement learning based in FSA. Several learning algorithms were implemented: Q-learning, PHC, WoLF and several combinations and slight modifications of that set. This reinforcement learning requires the ability to easily quantify feedback. The system provides only basic capabilities.

Nikovski worked in a domain of Decision-Theoretic Navigation of mobile robots. Nikovski's [52] work does not specify the particulars about the learning, but instead discusses autonomous agent movement and fundamentals of navigation. The paper discusses foundational (FSA) and Q-learning structures. Four different learning heuristics are explained, which attempt to resolve several key shortfalls in estimating states. Steep Gradient Ascent uses probabilities to predict the states. The Baum-Welch algorithm was adapted for Hidden Markov Model learning [52]. From these two algorithms, two novel approaches were proposed to overcome shortfalls. In Best First Model Merging (BFMM) the objective is to accurately predict which actual time-state pair the current POMDP observed state is modeling. Criteria for merging are used to decrease the likelihood of error in the states iteratively, to a decision appropriate level. State Merging (SMTC) searches for the same objective but instead of the greedy search used in BFMM, the system includes suboptimal solutions that can be merged in order to solve for an aggregate better solution in the long run. Tests are run on 5 planners using these learning techniques over the same data sets. None of the combinations converge to the optimum but the systems do show limited improvement. The cause of this non-convergence stems from the systems inability

to learn the correct model. Of those, Assumptive Planning with SMTC using vector matching prior to and after, combined with matching indicate better results. There also exists a bit of concern about mapping vectors leading into and out of a state if the system being modelled is Markovian. This work, although not thorough in the definition of the problem domain, presents several plausible solutions to the the UAV problem domain.

Roy uses a POMDP to move autonomous robots through a 2D space [65]. He uses this scenario to accomplish Markov Localization. The problem space is so large that the state is decomposed into belief planes. These belief planes allow the system to abstract the state, reducing problem dimensionality. The reduction technique used is Principal Component Analysis (PCA). It is an effective approach in the localization domain.

Khosla [41] use Evolutionary algorithms for Weapon Allocation and Scheduling (WAS). In this domain two parameters are optimized, Threat Kill Maximization (TKM) and Asset Survival Maximization (ASM). Three approaches are used to find solution sets. The deterministic selection evaluates the entire domain space and returns optimal solutions for pedagogical problems, but does not scale well. A GA proved more efficient and effective, solving to the sub-optimal levels outside the pedagogical limits. It also improved on the results from his other stochastic work. Finally the two were combined providing no improved results, again, stopping at pedagogical limits. Despite the fact that this mapping approaches more of a set covering problem than a POMDP [41], its mapping proves very insightful because it sets the tone for possible fitness objectives.

*2.1.1 POMDP Model and Complexity.* The POMDP structure is defined by the tuple in Equation 2.1.

$$D(S, A, T, O, R) \tag{2.1}$$

Here $S$ is the set of states, $A$ is the set of actions taken in $S$. $T$ defines a stochastic transaction from that state to the next. $O$ defines the set of agent observations and $R$ is the reward or feedback mechanism.

A POMDP of autonomous control vehicles relates to the sub-problem Vehicle Routing Problem (VRP) [1]. The VRP maps to the TSP a NP-hard Problem [5]. In Secomandi [70] the VRP with Stochastic Demands (VRPSD) is mapped to a MDP. There are a few differences in POMDP and VRP. First the VRP has a standard global knowledge of the system. Dynamic VRPs remove knowledge of all the nodes by letting the change [5]. Also, the VRP is deterministic, but the VRPSD proposed by [70] shows the constrained problem and connects it with the MDP. The mapping of the VRP by [5] facilitates comparison of the models:

- $v = v_0, , v_n$ is a vertex set, where:
  - Consider a depot to be located at $V_0$.
  - Let $v' = v\{v_0\}$ be used as the set of $n$ cities.
- $A = \{(v_i, v_j)/v_i, v_j\}$ exist in $V$; $i$ not equal to $j$ is an arc set.
- $C$ is a matrix of non-negative costs or distances $c_{ij}$ between customers $v_i$ and $v_j$.
- $d$ is a vector of the customer demands.
- $R_i$ is the route for vehicle .
- $m$ is the number or vehicles (all identical). One route is assigned to each vehicle.

The VRP does not isomorphically map to the POMDP. The state S is defined by the agents and targets or customers states in both cases. By focusing on the data structure of the VRP we can map it, the set of vertices, $v$, directly to the set of targets, $\tau$. The set of arcs, $A$, can be mapped to the set of SO action transition definitions,

[1]Vehicle Routing Problem is a benchmark optimization problem in which an agent has to make deliveries from a central location. Here the route is optimized for minimum distance, minimum time, or minimum service agents.

$T$. In the POMDP however the transitions are stochastic. The actions $A$ in both problems are movement between vertices, with a POMDP the movement is much more complex. The observations $O$ in the POMDP are shown by the changes in the environment through sensor readings which map to the changes to the environment in the Dynamic VRPs. If a reward for arriving at a customer is established, the VRP provides a simple $R$ based on state. The POMDP reward is a function of the current and previous state as well as the action. In the VRP problem there is only one agent but it is not bound by time so it can act like multiple agents. Optimization occurs on the weight of the arcs, or SO transitions, in both cases.

A VRP that is constrained to a given number of vehicles is NP-Complete. A TSP has a complexity of $O(\frac{1}{2}(n-1)!)$. Given a large number of vehicles the VRP, which is no more that a large number of TSPs, it approaches $O(n!)$ or NP-Complete [57, 75]. The POMDP complexity is larger than both cases, because of its expanded action, transaction, and reward sets. The complexity of the problem is loosely based on the number of agents, $n$, and the action, $m^2$ (movement in the map), and transaction possibilities, $t_p$, resulting in $O(n^{t_p m^2})$

Given the higher complexity, a deterministic POMDP solution is intractable for large n. With the combination of autonomous control structures, discussed in the next section, swarming with SO rules lend themselves well to solving problems in a dynamic, unknown environment. The ground work, for the emergent behaviors to be applied to this problem, is presented in Section 2.2.1.

*2.1.2 Interactive Partially Observable Markov Decision Process.* A subset of POMDP models focuses on the application of the Markov assumptions to independent agents with independent actions. There are three approaches to this subset of the model. The first defined by Bernstein [10], is the Decentralized POMDP. It specifically articulates actions and Observation sets for each individual agent. The second by Boutilier [12], called the Multiagent MDP, also specifically articulates the the set of actions for all agents. The Interactive POMDP (I-POMDP) used by Doshi [30]

specifically defines the state transitions of each agent based on the probability of the interactions with other agents. This approach focuses the agent actions based on its knowledge base and behavior set independent of the entirety of the domain.

Equation 3.8 shows the tuple defined by I-POMDP [30]. This approach focuses on decoupling agents acting in the same environment by adding belief of the effects of interaction to the state.

$$I - POMDP_i = \langle IS_i, A, T_i, \Omega_i, R_i \rangle \tag{2.2}$$

$IS_i$ defines the interactive effect of the agents on each others state through $IS_i = S \times \Theta_j$. In this the belief state of other agents $\Theta_j$ effecting the state derives from Equation 2.3.

$$\Theta_j = \langle b_j, A, \Omega_j, T_j, O_j, R_j, OC_j \rangle \tag{2.3}$$

Most of the pieces of this belief state derive from the POMDP but focus on the agent $j$. The $OC_j$ outlines the optimum criterion for the agents. This representation mirrors the actuality of the simulation and the stochastic nature of interaction and transitions. Further description of the I-POMDP and the proof of its mapping is seen in works by Doshi [30]. Through the POMDP the totality of the domain is represented and the I-POMDP pulls out the domain model of the individual agent and ties the two together.

## *2.2 Autonomous Control*

Autonomous control of vehicles transcends medium and motive, from underwater exploration to ground based safety to airborne military, the spectrum is large. As outlined in Figure 2.1, the levels of control vary as well. Despite all of these differences there exists an underlying control structure that can be utilized. All of the proposals in the following paragraph fall into level 7-10 of Figure 2.1.

| Automation Level | Automation Description |
|---|---|
| 1 | The computer offers no assistance: human must take all decision and actions. |
| 2 | The computer offers a complete set of decision/action alternatives, or |
| 3 | narrows the selection down to a few, or |
| 4 | suggests one alternative, and |
| 5 | executes that suggestion if the human approves, or |
| 6 | allows the human a restricted time to veto before automatic execution, or |
| 7 | executes automatically, then necessarily informs humans, and |
| 8 | informs the human only if asked, or |
| 9 | informs the human only if it, the computer, decides to. |
| 10 | The computer decides everything and acts autonomously, ignoring the human. |

Figure 2.1: Sheridan & Verplank (1978) Levels of Automation [72].

Gat [29] uses general approach focused on the most difficult aspect of autonomous control, stimuli response. In order to accomplish quick response in a dynamic environment, the system structure is decomposed into three parts. The *Controller* focuses solely on immediate response with little state knowledge. The *Deliberator* does all of the deliberate planning and the third communicates and coordinates between the first two, the *Sequencer*. Figure 2.2 shows this structure that is the widely accepted standard.

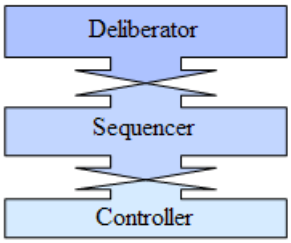


Figure 2.2: Gat Three Layer Architecture.

Rydsyk [66] uses a model that focuses on the level of knowledge: *world states, local states, vehicle states.* Through this model, he successfully implements autonomous UAV controls in a limited domain. Figure 2.3 shows his architecture.

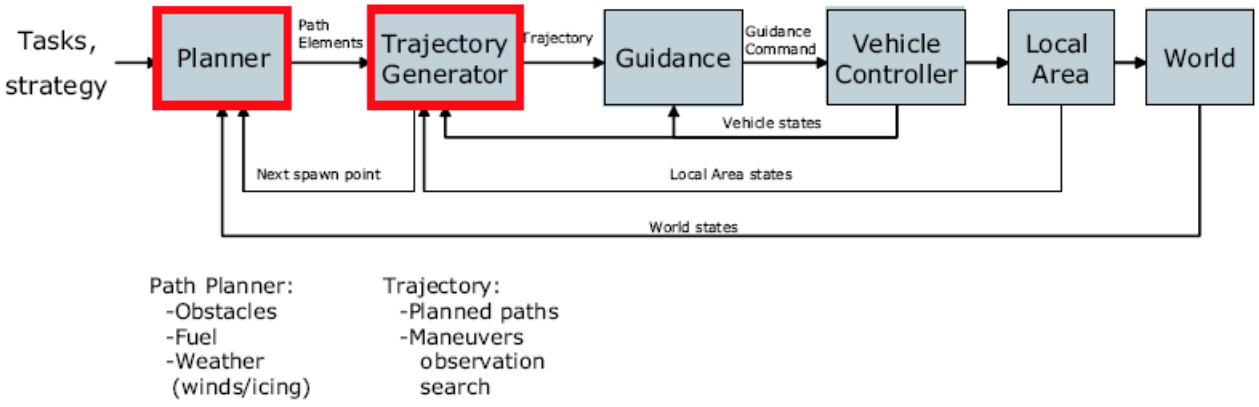


Figure 2.3: Rydsyk Three Layer Architecture.

Reynolds also [63] implements a structure to establish control for autonomous agents. In his "game" hierarchy, he establishes a three level structure: *Action Selection, Steering, and Locomotion.* Figure 2.4 shows the form of which he succesfully employs in game agents navigating game space.

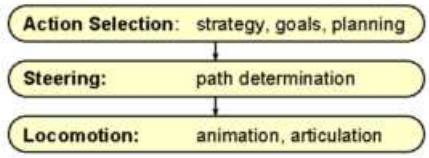


Figure 2.4: Reynolds Three Layer Architecture.

Rosenblatt [64] proposes a system that again separates the vehicle control from the learning, and is not so directly focused on the instantaneous response. His model has a set of modes that manages weight values for several control modules in the *Arbiter.* The values from the *Mode Manager* and the modules are used to calculate the next step and are then sent to the agent *Controller.* Figure 2.5 shows the architecture that he uses to develop successful ground based agents.

*Autonomous Control in SO Environments* In Swarmfare, [59] autonomous control is established through SO modelling. It combines the aforementioned structures into the three tier structure similar to that illustrated in Figure 2.6. The simulation revolves around the development of SO rules and the interplay of those rules. This


Figure 2.5: Rosenblatt's DAMN Architecture.

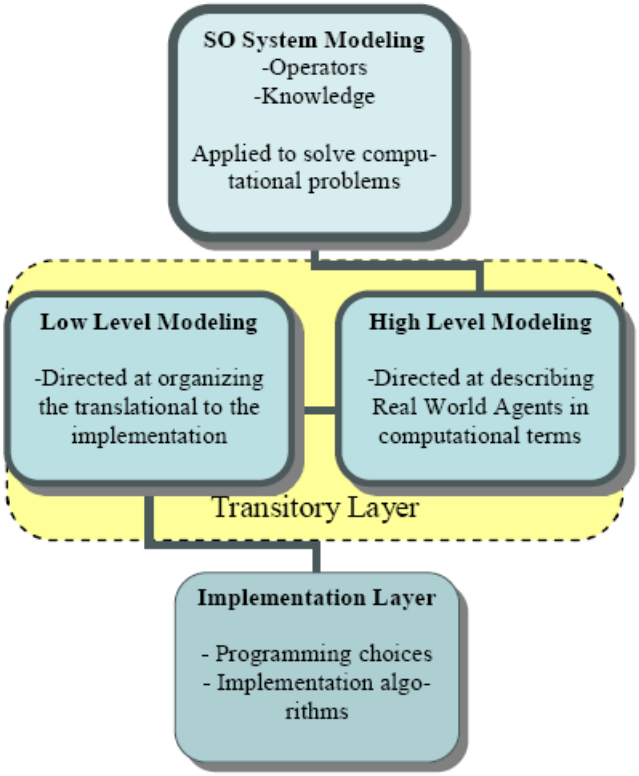


Figure 2.6: Three Layer SO system model [55].

takes away from the criticality of the immediate response ability. In Swarmfare, a three layer structure can be seen. The visualization represents the state of the swarm in the world in the UAVs operator, arbitration and control happen internal to the agent and the rules define reactive locomotion. That is mapped high to low as System State (the Swarm), UAV Agent State (Behavior Archetypes (BA)), and Update Local State through transition functions (SO rules). The first level maps the real world system to a set of knowledge and operators, this layer is called the SO System modelling. The second, transitory layer, consist of two co-dependent sub-layers. Translating the SO system model mapping into computational terms is the first sub-layer and the structure of the code from this architecture design is the second. Finally this design is mapped to code in the implementation layer. In the following section the ideas governing Self Organization are outlined and the specific rules used in Swarmfare are articulated.

*2.2.1 Self Organization.* The definitions of Self Organization are as varied as the people authoring them. Camazine defines Self Organization as [15]:

> "... a process in which patterns at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern."

Francis Heylighen sees SO as a sub-unit of his study in Cybernetics [34]:

"The spontaneous reduction of entropy in a dynamic system."

Cottam is wary of over use of the term and offers this description of SO [23]:

> "... transition from 'a set of components' to 'a unified system' is possibly the most fascinating aspect of our natural environment, especially as witnessed in the realm of (living) biological organisms This is, however, rather a slippery subject, as many, if not most reported examples of 'self organization' are primarily 'investigator-organized'. The emergence of new properties on changing level in a hierarchical assembly is often more attributable to un-noticed inter-level transformation of pre-imposed rules or initial experimental conditions than to self-organization."

| Category | Cazamine | Heylighen | Cottam |
|---|---|---|---|
| **Level of Origin** | Agent | Any | Component |
| **Level of Organization** | Global | System | System |
| **Environment Interaction** | Stimulus | Perturbation | Very little |
| **Operators** | Component Interaction | Subsystem Attractors | Inter-level Transformation |
| **Result** | Global Patterns | Reduced Entropy | Unified System |

Table 2.2: SO Table.

These definitions are placed side by side for comparison in Table 2.2.

In decomposing the definition, a SO system's order emerges only through the interaction of it agents. These systems are made up of a group of agents, usually referred to as a swarm. This swarm achieves a stable physical or biological structure that produces a desired effect better than any single agent could achieve on its own [59]. This desired effect manifests itself through a positive feedback loop that perpetuates not only the action, but also the existence of the system. Generally speaking, SO evolves by the interaction of the agents based on their inherent properties, internal knowledge base and the communication between agents.

Agents in SO systems can be described as automata [59]. Merriam Webster's defines an automaton as a "mechanism that is relatively self-operating." [27] As part of SO systems, all agents must be autonomous. Every agent must function completely on its own. It must be able to make decisions, react to its environment and communicate on its own. If the agent does not fulfill all of these requirements, the system is probably not SO.

To be considered SO, it is only the agent's interaction with other agents, the environment, and other systems that creates organization. In most cases, when the results of this interaction are viewed from a macroscopic level the order is revealed. This order is a function of the simple rules inside the agent (or subsystems of agents) working together to achieve an overall desired effect. The synergistic interplay between these systems is what prompted Haken to coin the theory of Synergetics [32]. He

defines the interdisciplinary idea of Synergetics when systems form, at a marcoscopic level equilibrium, not at thermodynamic equilibrium, in an open system based on SO characteristics. It is impossible to define Synergetics without Self-Organization and vice versa.

Formation in SO systems is a bottom up process. The agents at the lower levels interact to create the organization. The feedback, however, is at a higher level [23]. For instance, if a few birds suddenly fall out of formation in a flying V, the entire flock feels the effects. The birds ahead and behind all feel increased drag, forcing the formation to reorganize. Feedback at the global level is what reinforces operators and parameters at the agent level to form the system's structure.

*2.2.2 Improperly defined SO.* Sometimes systems are mistakenly defined as SO. Listed here are several other strategies (with examples) that may be employed to create organization in a system but not nesscarily SO [15]:

- Leadership/Hierarchies : Lion Prides
- Blueprints : Human Construction
- Recipes : Spider Webs
- Templates : Male Villager Weaverbird (uses its own body as a template when constructing a nest) [42]

All of the strategies listed focus on the use of global knowledge by one or all of the agents in the system. Any global knowledge of the environment or systems by an agent disqualifies the system from the Self Organization classification. Again the defining property of SO is the dynamic interaction of the agent level characteristics and rules that result in emergent properties or behaviors [15].

*2.2.3 SO System Strong Casting.* Agents in real world SO systems find themselves forcibly tied to a system once it has been created. In most cases the cooperation of the agents in the system has a multiplying effect on the capabilities

needed for survival of a single agent. A single agent is far more likely to be successful as part of the system as opposed to going it alone. Of course, this phenomenon does have its disadvantages. Being tied to a system, which exploits capabilities that are drawn out at a global level, forces the system and the agent into a niche for survival. Male fireflies for example use the aggregation of hundreds of flashes to communicate during mating season. If the light flashes were gone, then the combined effectiveness of the system would also disappear. If each agent had to fend for itself, the system as a whole would terminate. This is strong casting, when agents in a system cannot adapt to a change in the environment or system, because the system is evolved to fit a niche. If a system were weak casting it would not force the focused development of a subset of the agent's operators, and thereby prevents the emergence of structure and organization. The effects of casting can be applied when looking for forward mutation of the system as well [35].

When agents are forced to accomplish the tasks because they are imperative to the system's survival, the agent has little flexibility to continue to evolve. This essentially creates a glass ceiling for the agents' capabilities. Interdependence causes a "free lunch" syndrome [23], where the emerging patterns in the high layers remove some of the autonomy for the agent.

*2.2.4 Characteristics and Classification of SO Systems.* The human mind, attempts to find patterns and characteristics that help classify things in its environment. Accordingly many characteristics have been developed to classify different types of SO systems.

It has been surmised that SO occurs in all realms of science [35]. In physics the phenomenon is pervasive from crystallization to sand wave patterns. In chemistry, SO is formed in chemical reaction times and chemical structuring. There have been many published works with biological examples ranging from bird formations to ant colonies [15]. In human society, organization has permeated throughout population centers usually unbeknownst to the inhabitants. John Holland claims that SO principles in

economics and logistics are what keep cities running. [36] The range of applications in scientific disciplines is seemingly unbounded, including the computational world.

*Hierarchy* Exploring the hierarchical structure of organization in a system helps facilitate the way in which problems can be decomposed and solved using SO systems. Hierarchy describes the level at which patterns/organization are observable in SO systems: internal to an agent, sub-system, and global, shown in Table 2.3.

| Internal | Sub System | Global |
|---|---|---|
| Intra-Agent | Inter-Agent | Inter-Agent |
| Visual Pattern (Geometric) | Immediate Response | Structures and Repeatable Behaviors |
| Temporary | Temporary | Long Term |
| Single Emergent Property | Single Emergent Property | Several Emergent Properties |
| Homogeneous | Homogeneous | Heterogeneous |
| Animal Coat Patterns | Ant Bridges | Termite Mounds |

Table 2.3: SO Classification Levels Characteristics (Smallest to Largest)

There are many examples of SO occurring internal to a single organism. Zebra stripes are one notable example which is produced at a cellular level through chemical interactions [15]. Other systems develop organization in a subset of a swarm. For example, in the insect world, a bee hive's response to aggression is not carried out by a single agent or by the entire swarm, but by some subset of the entire swarm. A large majority of SO systems, however, produce global organization through simple interactions at the individual agent level. This type of interaction ranges in complexity depending on the agent. It can range from simple mold colonies, shown in Figure 2.7 to the herding characteristics of large mammals. The slime mold's sole emergent property is the assembly of large numbers of agents for feeding. Ants or termite swarms can have multiple emergent properties including feeding, construction, and breeding. [15] Again, in global SO the agent does not have knowledge of the global situation, only of its local surroundings. Global properties emerge independent of the

agent's knowledge. This first and most fundamental characteristic impacts how all of the other characteristics affect the agent and its swarm SO behavior.

Figure 2.7: Mold Growth is defined by SO principles

*Decision Making* The rules of a system are based on its current state and the introduction of stimuli from the environment or other agents in the swarm. The agent or system is stable until it reaches a decision point. When it passes a threshold, an agent modifies its behavior based on a drastic change in environment. The response at these thresholds can follow several different predictability models. Many systems are driven by a simple deterministic function. In this case, the response is predictable and based linearly on the action of the adjacent agents. For example, in fish schools, shown in Figure 2.8, the vector of an agent is defined by the adjacent agent's vector [15]. Other systems take a stochastic approach where agents make decisions based on the probability of a random variable. An example of this is illustrated in the ant world. When an agent reaches a trail intersection, which pheromone trail they follow is based on a random variable. This allows for variation in their foraging patterns. There are several other possible variations on the predictability of the response but these are the most common.

*Heterogeneous vs. Homogeneous Swarms* The appearance of different agents in the same swarm can also affect the way that SO is achieved. The tradeoffs between homogeneous, all the same type agent, and heterogeneous, multiple agent types swarms


Figure 2.8: Schooling fish (photograph provided by Brandon Cole [21])

are not always clear. In most cases in nature, a homogeneous group has a simpler rule set, but at the same time, cannot accomplish as large a variety of tasks. The complexity of heterogeneous swarms provides more capabilities with some costs, but through the implementation of simple guiding rules, self organization is still relatively easily accomplished. The most studied example of a heterogeneous swarm is the ant hill. There are some species of ants that have a half dozen or more different subtypes. Subtypes range from the well known foragers and queen to agents with very specific jobs. The "majors" in ant hills are usually large with over grown mandibles and are capable of suicidal attacks to any nest intruders [11, 15]. The roles of the agents in a heterogeneous swarm are extremely malleable. If a swarm loses most of its foraging types, other types such as the "majors" can pickup that work until the forager levels are replenished. This distribution of work is, in itself, SO and seems to develop given the situation. Once again all of the organization in this system is a response on the individual level to the state it perceives in order to form emergent system structure or behaviors.

*Motives* Another defining characteristic of SO systems is the driving force for the organization. Many groups of biological agents have developed SO properties through applications that allowed the swarm to thrive. In many cases, the control and exploitation of resources is at least one of the factors that drive SO. There are many other motives for SO behavior including mating, defense, efficiency of move-

ment, and communication [15]. Several SO systems seem to have multiple reasons for their organization. Whether that is by design or is merely a side effect is difficult to determine, because most systems can not be isolated for long enough times to determine the origin of organization. The impetus for a system's SO can, however, be a helpful indicator when looking for a SO system to use as a biological model for computational problems.

This list of characteristics includes only a few of the largest and most prevalent of SO characteristics. As the knowledge base of SO systems increases, the need to classify the system by different characteristics causes this list to grow.

*2.2.5 Operators Introduced by Biological SO Agents.* Biological SO systems and swarms have many successful applications to computational models. High level operators, which have manifested themselves in the biological world, can be modelled and exploited in the computational world. The generic goal is to obtain an understanding of how the biological agents interact in order to develop a tool box of operators for solving computational problems.

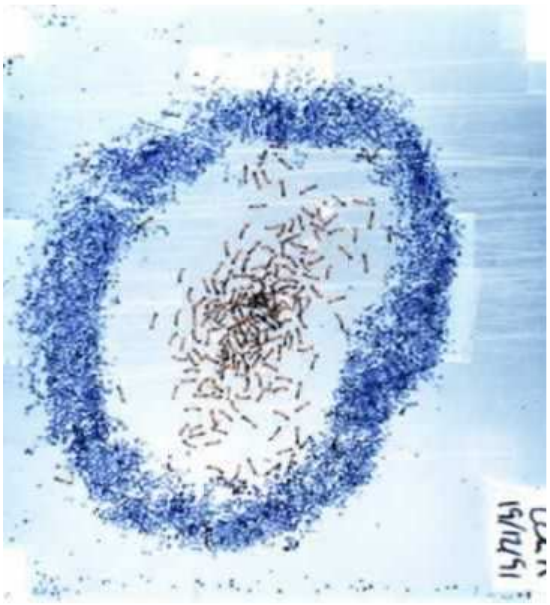

Figure 2.9: Stigmergy (Ant wall construction) [15]

*Stigmergy* Stigmergy is an operator that is pervasive in SO systems [15]. It is defined as an action that is taken by one agent that indirectly communicates information to another agent. When a builder ant places a stone based on where the last agent placed a stone, it is stigmergy. Over time the repetition of these actions results in the construction of walls. Most SO systems use some kind of stigmergy, but communication, through directed signals or contact, is also prevalent. The dynamic between the two is very indicative of how a system is self-organized.

Figure 2.10: Pheromone Trail (Leaf Cutter Ants) [15]

*Pheromone Trail* The most well known SO operator is the ant pheromone trail and a subset of stigmergy. In this system, the ant leaves a scent trail by rubbing its lower abdomen against the ground as it returns from an active resource to its nest. Over time, several agents repeat this operation. As the pheromone aggregates on that trail other ants are assisted in navigation to the goal. This is similar to many other animal scent markings which designate trials or territories. In the digital world there are no chemical pheromones, but agents can explicitly communicate information on successful paths. This operator has been extensively used in the computational world for the Travelling Salesman Problem and other optimization problems. [11, 59]

*Clustering* The clustering or classification of objects is developed by SO systems to facilitate expeditious organization of its environment. It too is a specific stigmergy operator. In ant colonies an individual agent can quickly determine and process objects. Objects ranging from brood in different stages, to expired agents, to waste or any other item seemingly out of place, can quickly be identified and moved to their proper location [11]. This classification operator is highly desirable in computational models. Humans also classify objects in order to gain a better understanding of their environment.

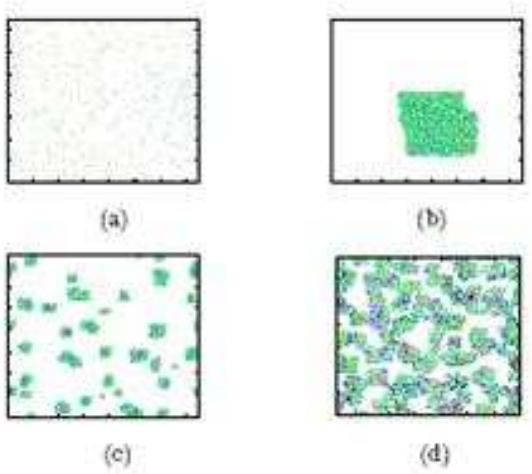


Figure 2.11: Evolution of Agent Hierarchy [76] (Red, Green, Blue - Highest to lowest class)

*Schooling* Another common operator is relative vector matching, like schooling of fish and flocks of birds which keeps large groups together through coordinated movement. Given the inherent danger of close maneuvers in swarms, this operator is extremely beneficial and has proven fairly easy to implement. Several models have been created but the most prevalent one was created by Reynolds. Figure 2.12 shows the three simple swarming rules: cohesion, collision avoidance (separation) and vector matching (alignment) [62]. As the agent reacts to all three of these rules at once, it quickly finds a balance of distance, speed, and direction that avoids collision and departure from the swarm.

Reynolds developed extended models that allow for automaton reactive response to environment stimulus. In his paper "Steering Behaviors For Autonomous Characters," [63] he addresses the broad scope of automaton behavior focused on agent or swarm reaction to other outside agents. He developed the following capabilities expanding on his three rules of cohesion, collision avoidance and vector matching: seek and flee, pursuit, evasion, offset pursuit, arrival, obstacle avoidance, wander, path following, wall following, containment, flow field following, unaligned collision avoidance, flocking, and leader following [63]. Of these, three capabilities, seek and flee, obstacle avoidance, and wander, form the building blocks for the remaining of

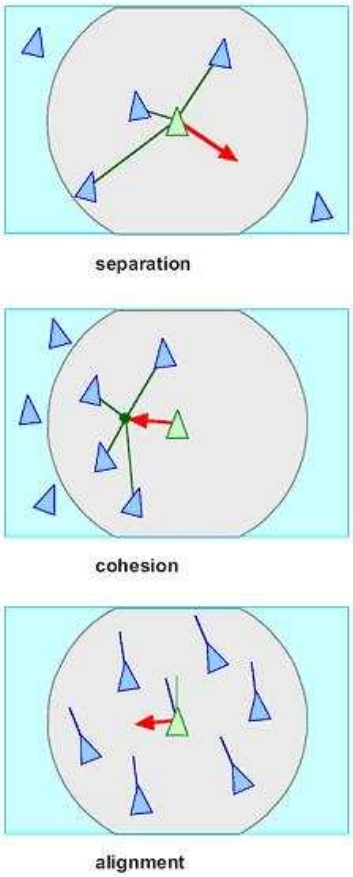


Figure 2.12: Reynolds Swarming Parameters [62]

the capabilities.

*Evolution of SO System Internal Hierarchy* Another operator that is extremely intriguing is the evolution of hierarchy amongst the agents of a system. In the summer of 2006 Tsujiguchi and Odagaki [76] built a society from "equal" agents through a challenge based process. When two agents would meet, they would 'fight', compare parameters, and the winner would gain a point, while the loser would lose a point and move near the winner. Over time hierarchies and population centers started to emerge. This could be useful in setting up ad-hoc networks, redundancy, or command, by making the hierarchy self-adaptive given a change in the environment.

This discussion is a survey of several prominent operators in order to establish a fundamental understanding of how SO looks and operates. Most of them have useful applications to UAVs and swarming simulations. A larger survey of SO operators is discussed in Appendix A.

*2.2.6 Existing Structure.* As this research effort is an extension of previous AFIT work, we briefly illustrate the existing structure [55]. Price's [59] focus when building the foundation of Swarmfare was to establish a solid Self Organizational structure for the system while creating a simulation that could still reasonably emulate UAV kinematics and communications. He developed this system as a distributed java system, that uses Genetic Algorithm to accomplish weighting of the SO rule set.

*Top-level SO model* Although Price [59] used the term "High Level Design" to mark the beginning of his mapping from real-world SO to the computational world, the top level is still present. In the case of Swarmfare, the SO system model most closely resembles a flock of birds. The basic structure derives from Reynolds [62]. This SO model provides a knowledge base and many operators, including inter-agent detection, the three functions of flocking, cohesion, collision avoidance and vector matching. Also, Price includes goal seeking, sensor interpretation, pheromone based attack operators, and unidirectional interagent communication.

*Behavior Archetypes* When SO is applied to a system, rules must be combined in order for an agent or computer to accomplish a coherent response. In the case of Swarmfare, the system gathers these rules together to form Behavior Archetypes (BA). Through these groupings the rules are weighted and applied to establish each subsequent action. This is similar to the modes in Rosenblatt's architecture Section 2.5. Through this open scheme of BAs, the system could theoretically be loaded with many different schemas that would optimize the reaction to a situation at any given point. For instance, currently there exists a BA focused on the rudimentary accomplishment of searches. This BA allows for the maintenance of looser group cohesion in order to maximize the swarm's collective sensor footprint.

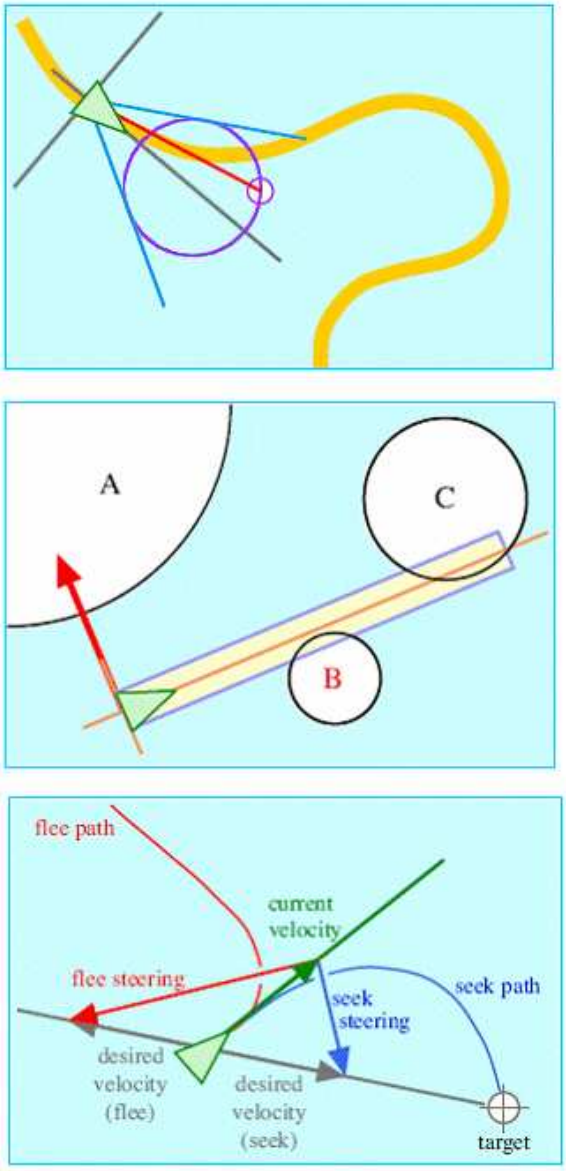


Figure 2.13: Reynolds second generational rules. Top: Wander, Middle: Obstacle Avoid, Bottom: Seek and Flee. [63]

*Rules Utilized* Thus far in the Swarmfare simulation, the system combines 10 rules to define the BAs:

- Flat Align - vector align with neighbors
- Target Orbit - orbit target at safe distance
- Cluster range towards - cohesion
- Cluster range away - separation
- Attract - towards center of mass of all targets
- Weighted Attract - towards closest target
- Target Repel - repel if with 90
- Weighted Target Repel - repulsion based on proximity to target
- Evade - a priori collision detection and avoidance
- Obstacle Avoidance

The ten rules are derived from the two BAs, swarm and target interaction. The swarm rules, flat align, separation, cohesion, obstacle avoidance and evade come from Reynolds work [62, 63]. The target driven rules, attract (and weighted), repel (and weighted), and orbit are derived. Attract and repel tries to form a balance of aggression and respect towards targets. The orbit stems from Lua's work [47]. Of these each one is weighted differently depending on the makeup defined by Behavior Archetypes of the agent. Appendix C shows a more extensive description of the behavior set from [59].

## *2.3 Other Simulators*

In this section various works in UAV simulation are evaluated. The impetus for evaluating UAV simulators is to choose the right tools to apply to UAV Swarm Intelligence. A series of sub goals that must be addressed when evaluating these systems includes:

1. Swarm Behavior Characteristics Modelling Capabilities:
    - Schooling (cohesion, collision avoidance, and vector matching)
    - Behavior morphism
    - Dynamic communication architecture
2. Modular Coded:
    - Allows for insertion of new modules/capabilities
3. High Performance Computing Traits:
    - Evolutionary Algorithms
    - Multi-Objective analysis
4. Data Collection and Statistical Analysis
5. Plug and Play Kinematics
    - The system needs a solid kinematics backbone (minimizing the effort needed to create accuracy in this aspect)
6. Visualization
    - 2D or 3D visualization (3-D preferred)
    - High fidelity graphics are not necessary

in the following subsection, a list of simulators is briefly described in reference to this list of goals. In Appendix B, a more through chart compares the simulators.

*2.3.1 Languages.* When establishing the right simulator to use for this type of research, the language or backbone architecture involved can play an extremely decisive role. The language must be able to do computing at a relatively quick pace because the computations are extremely large. It must have a structure that allows for control and manipulation of the low level system. As a juxtaposition the language

must not be low level itself because more time would be spent setting up the problem than building and testing the actual objectives.

MATLAB is able to accomplish the heavy mathematics to emulate real-world UAV flight, but the overhead to run such a detailed analysis may not be worth the effort for this type of research. The C family of languages is an attractive option because of its ability to control low levels of the system. Some of the C type languages make this rather difficult, but Java allows for access to those system level calls but does not require the extensive programming to make them operate well. Very few other languages currently offer any extensive knowledge base in the SO UAV research area.

*2.3.2 Candidates. Self Organization (SO) Simulators* This group of simulators is characterized by a focus on pure self organization. The majority have only fundamental interactions rules from which the program/researcher can draw out self organization. Overall the SO simulators capabilities are extremely similar. Advantages as a set include: ease of use, plethora of examples, and conformity to SO ideals. Disadvantages include: lack of UAV support, no high performance computing and constraints on interagent communications.

The first of these SO simulators is SWarm Evaluation and Experimentation Platform (SWEEP). It is one of the first generation SO simulators developed by Case Western University. It has average modelling, graphics capabilities, and analysis capabilities. It is open source Java. This simulator has been replaced by many of the more robust SO simulators space.[9]

The next simulator is Multi-Agent Simulator Of Networks (MASON), developed at George Mason University. It has a large library of examples and has been used across departments at that school, including economics and political science. The analytical capabilities are a bit clumsy, it has no Multi-Objective Evolutionary Algorithms (MOEA) capabilities, only basic visualization and very little control of kinematics, but overall it is a good SO tool. [9]

The SWARM code managed by the Sante Fe Institute is one of the most well known simulators. Due to the open source nature of the Swarm software and its support through SFI, it has the most robust modelling capabilities. The visualization is only average, with basic kinematics and 2D or 3D rendering. Overall it is a very capable simulator for all generic SWARM and SO functions. [9]

Recursive Porus Agent Simulation Toolkit (REPAST) built by Argonne National Laboratory. is one of the best SO simulation tools. This particular tool has a large library of existing code, good graphics, and average modelling capabilities and the best analysis package, although it has never been used for in-depth UAV swarm simulation. [9]

*UAV Swarm Simulators* This group tries to blend the UAV kinematics and simulations with SO and swarm modelling. The difficulty is finding a balance between the aforementioned objectives.

The MATLAB organization sponsors a project called MutliUAV. It is a basic model that combines the basic kinematics with some swarm functionality. Its capabilities were not fully flushed out because of its reliance on MATLAB licensed plug-ins not held by AFIT. [3]

Another system called MultiUAV sponsored by Air Force Research Labs (AFRL) is also a capable tool. It has an UAV support architecture that closely represents the real world. It is also based in MATLAB. The controls, kinematics, and communications capabilities are packaged closely approximating the realities of current UAV flight. At this time the simulator is capable of swarming 8 UAVs. The present swarm capabilities diverge from some of the fundamentals of SO; primarily this is because it forces top down swarm behavior as opposed to emergent SO properties. The agents in this simulation do still act as a cohesive unit. The simulation is capable of producing a 2D visualization with targets and some basic terrain. It is also capable of collecting a reasonable amount of analytical data. Overall this simulator/modeler is very capable if given smooth scalability to over 8 UAVs is possible. The biggest disadvantage

is its accuracy to SO. To program in SO capabilities at a high level, a tremendous amount of effort must be made to understand and apply the lower level individual UAV characteristics.

[7] Simulation by IcoSystems - the corporation headed by Eric Bonabeau - is an older simulation. IcoSystems no longer supports it nor is it advertised on the corporation's website. AFRL used to work with it but they have not used it in their research in several years so very little information about its capabilities exists. [10]

The most readily available simulator is SwarmFare developed in 2006 at the Air Force Institute of Technology by Ian Price. Built in Java, the SwarmFare software uses GA to accomplish swarm cohesion and basic search and destroy capabilities. The software produces sequences given a bounded environment with targets. There is also a visualization package that reads the output scripts. This script is a by-product produced during the evolution of the chromosomes, from the GA. It can be used to create a 2D animated display. The simulation is appealing because it is built to allow the appropriate level of kinematics and communication without losing focus on the true nature of the research - swarm SO. [6]

## *2.4 Parallel Computation*

In order to execute the complex computing involved in MOEAs a high performance computing framework should be used. Currently Swarmfare implements two different parallel GA schema, farming and island models. In order to further parallelize and thereby decrease the simulation time a Parallel Discrete Event Simulator (PDES) could be used. Appendix B.2 shows several of the PDES' available.

SPEEDES a PDES by Metron has been used by many past researchers at AFIT. It is written in C. This system provides an event simulator construct that can accomplish distributed computing for multiple agents. To do this, each agent's actions can be computed independently and, if agents effect each other's logical paths, the SPEEDES environment can back track through saved states to recompute a new path.

The capabilities of this tool is largely negated by the SwarmFare code's capability to accomplish its own parallel computing. [1] [6] The SPEEDES would be applied to Swarmfare by parallelizing each action of each agent. Because of the constant interaction of the agents this may not likely be an effective approach to reduce computation time.

## *2.5 EA approaches*

*2.5.1 Why a GA?* Correct application of search techniques find solutions in the problem space that create functional swarms. Because of the size and complexity of this problem domain stochastic search techniques must be utilized. There are three main members of stochastic Evolutionary Algorithms class: Evolutionary Strategy, Genetic Programming and Genetic Algorithms. In general all of these techniques take a population, evaluate the individuals, evolve them through mutation operators, and perform a selection on the next generation. EAs use simple primitive data structures vector as chromosomes which provide the ability to guide searches and optimize solutions. Evolutionary Strategy use a data structure of real value vectors and strategy parameters. It relies heavily on an understanding of the phenotype space, which is not present in this problem domain. Research in GPs, which use tree structures, showed success in facilitating growth over structure control systems [8, 50, 58, 83]. The initial work of applying GAs to SO swarms proved successful in [59].

Other stochastic algorithms have been explored to solving this problem by Nowak in [54]. Work using Ant Colony Optimization (ACO) only extends to the routing of the agents and requires another control structure to facilitate the swarming and other behaviors [4]. It also shows an application where a Particle Swarm Optimization (PSO) works on a similar data structure, however no works show successful results in this problem domain [37, 56]. Finally, Artificial Immune Systems (AIS) seems to fit better as a repair function than the results shown in autonomous control work [43]. Because of the complexity of this domain deterministic solutions

are unfeasible. Other stochastic biologically inspired systems have not performed well for large dimensional search spaces.

For these reasons this paper expands the research into current problem by extending the GA domain into the Multi-Objective Optimization Problem (MOP) realm and solving with a Multi-Objective Evolutionary Algorithm (MOEA)

*2.5.2 Chromosome.* First we must outline the data structure for the GA. Formulation of the chromosomes derives from the objective functions. In order to reach the objective of target engagement, the system must produce emergent behaviors that aggregate the capabilities of the UAV in a swarm. The EA must control and integrate the SO rule sets to form this emergent behavior. Therefore the mapping of the chromosomes relate to the control parameters or weighting of the rules sets. Figure 5.4 shows the make up of the chromosome structure.

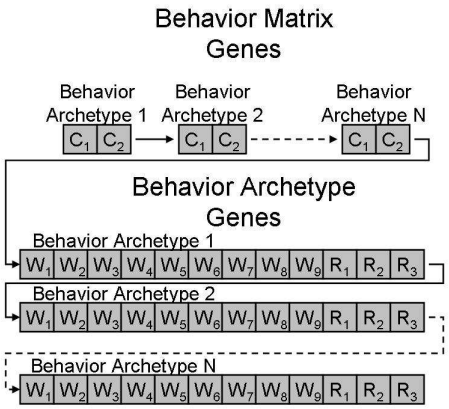


Figure 2.14: There is a connection weight for each sense for each behavior archetype. These are followed by 12 genes which describe the weights and radii for the behavior rules for each behavior archetype.

*Control*

Implementing the transactions defined by the behavior set in Section 2.2.6 requires synergistic integration of those behaviors. Here behaviors are vector fields that direct the agents movement, similar to that seen in [33]. The arbitration between these behaviors needs to utilize the simplified state that the SO behaviors react to

themselves. For this reason a simple multi-layer perception facilitates the differences between the state sets. The control weightings for this structure are also included with the behavior set weights in the chromosome shown in Figure 5.4. Here each control weighting and corresponding behavior weight set form an adjacent sub-string in the chromosomes.

Given the multiplicity of states in an environment, multiple sets of rules or modes direct the swarm, shown in [59]. Therefore a control structure must be used to change the mode. In this simulation environment a network of preceptrons senses the current environment and does the arbitration between modes or Behavior Archetypes (BA). These BAs allow the system to develop several sets dynamically of weightings in order to react to different situations. For example a chromosome with three BAs and 9 rules will have 42 alleles. Several of these sets form the full structure of the chromosome.

Figure 5.4 shows the representation used. The chromosome values are used to map the weightings of each rule. Note that the evolutionary operators work on the bit level, so in order to translate the chromosome values Gray coding must be exploited. With Gray code the system minimizes the effects of those operators, because the change of a single bit will only change the value of that gene by one as well.

### *2.6 Background Summary*

In the subsequent chapters of this thesis document these three areas, POMDP, Autonomous Control, and GAs, are tied together seamlessly. In doing so the emergent behaviors of a SO swarm created by the rule set engage targets in an autonomous manner. We build on the SO sandbox first developed by Price in 2005-06 [59]. An added Graphical User Interface, extended parallel schemes and redesigned implementation using software engineering principles allow for exploration of SO rule sets to "optimize" target engagement effectiveness and improve computation efficiency.

# III. Problem Decomposition

This chapter defines the high level design starting with the problem domain and mapping it to the algorithm domain. In order to accomplish this the generic Partially Observable Markov Decision Process (POMDP) model must be adapted to fit this problem domain. The focus of this POMDP mapping effort centers around the more specific objectives related to optimized engagement with targets. The problem domain maps to a SO decomposition and from that a set of SO rules are developed that lead to the desired emergent behavior. In order to develop the needed control weightings for the control structure, all controlling parameters are mapped to a chromosome and MOEA algorithm.

The structure of this section follows from the illustration of the problem domain to the approaches used to optimize in that space. This includes a verbose and mathematical model of the problem domain along with discussion of verifiability and validity of the system. The mapping of the problem domain is then translated to the set of SO rules specific to target location and engagement. From that the emergent control structure technique is described. This chapter concludes with the application of the GA for optimization in this space.

## *3.1 Problem Domain*

From the problem domain notionally described in the Chapters I and II, we decompose all aspects of the problem. We use the U-decomposition structure described in [53]. The definition of the problem domain is described in detail. The step-down approach structures the problem into subproblems leads to a set of operators. They include the movement definition, the agent's governing parameters, explanation of the terms of the hostile environment, and all other constraints. As the structure is built from the bottom up the system evolves into desired emergent behaviors. From this structure the interaction of the GA with the SO is defined. The construction of the appropriate GA is a result of that interaction analysis.

*3.1.1 Verbose Problem Description.* The problem of effectively engaging targets, given effective swarm capability, revolves around two main components. First, the targets must be found. In modern conflict, the practice of predicting the general area in which masses of force and targets are located is reasonably accurate. The exact locations are more difficult to pinpoint. The second aspect, engagement of targets once found, takes skill and coordination. The decomposition of these two points is the focus of this section.

*Getting to the Threat Area* As noted, the location of Threat Areas (TA) is assumed known. The first step is getting to those areas. The existing SO swarm capabilities as defined by [59] create a dynamic swarm capable of moving through an environment. The next step is adding a path that can work through a domain space to relative TA follows. Once a path is established through the domain, the addition of SO rules for migration [24], or path follow [63] accomplish this task. Once in the target area the system can transition into the target engagement mode.

*Choosing and Engaging Targets* The introduction of targets dramatically changes the desired behaviors of the swarm. Here the swarm must address the problem of target yield versus danger. Attaining this information requires localized reconnaissance. Full mapping of the space surrounding the target minimizes the uncertainty of a decision process. Based on the gained knowledge of the TA the individual agents establish target priorities and as the space changes due to loss of agents and targets so too does the risk and reward of the remaining targets in the area. From that, the swarm engages targets maximizing the success rate by assuring the targets are attacked with adequate force. Bee swarms perform a similar reconnaissance and decision sequence during the movement to new hive locations. [51, 71, 79]

*Target Engagement Techniques*

Previously there were two approaches that could be used in the target engagement and sequencing, those shown in Lua's work [47] or modeling on current Tactics, Techniques and Procedures (TTP). Lua's work develops a scenario in which the

agents line up circling a target and peel off to engage the target. This effectively solves the problem of traffic in a localized area around the target and coordinates movements locally. It, however, does not address the way in which targets are chosen nor does it utilize all the Principles of War as outline in Air Force Doctrine Document 1 (AFDD1) [39]: Unity of Command, Objective, Offensive, Mass, Maneuver, Economy of Force, Security, Sunrise and Simplicity. In Lua, only Surprise and Objective are developed.

TTPs present their own set of difficulties. Different units with different missions develop different TTPs. TTPs do not transcend the the domain of war like the principles, therefore are not agreed upon across units. Also, as war changes over time so to do the TTPs. Finally working TTPs into an SO swarm would amount to scripting. With SO emergent properties that give way to dynamic behaviors that can react to any situation, these are much more applicable. For this reason we look into target selection and engagement procedures as seen in the natural world.

*3.1.2 Mapping to a POMDP.* Mapping the UAV Swarm problem to a well defined problem domain has two advantages. One, in the construction of the UAV swarm domain the problem statement can be thoroughly instantiated. Two, standard problems may have libraries of research techniques for solutions in a given domain. For these reasons we map the verbose problem description to the problem domain of POMDP.

The UAV problem mapping to the Partially Observable Markov Decision Process (POMDP) domain requires explanation of several critical elements. First the targets and UAV (or agents) have sensors and interact through basic nearest neighbor communications, with epidemic transfer of information. As a result any agent in the space has only limited knowledge of its circumstances. As the UAV agents move through the space, engagement with the targets follows stochastic modelling and allows for aggregation of forces. This creates a scenario where the domain space changes rapidly and unpredictably. The agents are not able to predict the transition

based on their action either. To reduce the confusion and computational complexity, the Markov assumption is used. The system works on an abstracted state because it does not have global knowledge or the inability to predict the results of any single universal action. Such techniques in state abstraction have been used before in [65].

The family of Markovian Models contains four primary members, Hidden Markov Models, Markov Decision Processes, Markov Chains and POMDPs. The UAV Swarm problem domain has stochastic control over the transitions between states and does not have global state visibility. For these reasons the problem domain falls under the category of POMDPs, see 2.1, [45]. The expansive problem domain space of POMDPs forces the reduction of the state into abstract pseudo-states. This facilitates ease of understanding to the developer. They also provide less computational complexity. Since the problem has been described in words we extend the mapping of the specific UAV Target engagement problem to a POMDP. This requires redefining and expounding on the basic POMDP structures, from section 2.1, shown in Equation 3.1.

*POMDP State*

$$D(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{R}) \tag{3.1}$$

*State* Equation 3.2 defines the state tuple for D:

$$\mathcal{S}(\upsilon, \tau, \zeta) \tag{3.2}$$

Within each state $\mathcal{S}$ there exists a set of agents $\upsilon$, set of targets $\tau$ and set of obstacles $\zeta$ in the domain space.

*Agent*

The agent (UAVs) is defined by the tuple in Equation 3.3

$$\upsilon(\lambda_{\upsilon t}, e_{\upsilon}, d_{\upsilon}) \tag{3.3}$$

Here the $\lambda_{\upsilon t}$ defines the location and velocity vector, $e_\upsilon$ is the engagement range of agents. $d_\upsilon$ is the detection range of agents. This state describes the agent from the global view, the agent state itself as a local view is extended with the I-POMDP model in transition section 3.1.2.

*Target*

Equation 3.4 defines the target (SAM site) sets:

$$\tau(\lambda_\tau, e_\upsilon, d_\upsilon) \tag{3.4}$$

Again the $\lambda_\tau$ defines the location vector, $e_\upsilon$ is the engagement range of targets, and $d_\upsilon$ is the detection range of agents.

*Obstacle*

Equation 3.5 defines the obstacle set:

$$\zeta(\lambda_\zeta, s_\zeta) \tag{3.5}$$

Again the $\lambda_\tau$ defines the location vector and $s_\zeta$ defines repulsion field strength of the obstacle.

*Action*

The action set is defined by the agents actions Equation 3.6:

$$\{\mu_{A_1}, \ldots, \mu_{A_n}\} \epsilon \mathcal{A}(\mu) \tag{3.6}$$

The $\mu_{A_k}$ is the movement action of all agents taken in the domain. The null action does not exist in the set as the agent must always be moving while in flight. This also means velocity vector $\lambda$ must not be 0. The set also includes other actions such as, detect, engage, turning and others depending on the capabilities of the agents.

*Transition Set*

The transition in the world domain is dependent on the independent agent action and interactions. Therefore Equation 3.7 show the transition probability between states.

$$T : P(s') = P(s', \Upsilon, s)P(s) \tag{3.7}$$

The probability of entrance into a new state $P(s')$ is dependent on the previous state $P(s)$ and the set of agent actions, $\Upsilon$. The agent actions are defined by the set of beliefs state transitions and the scope of the set defined by the following. The agents local state and actions can be defined by the I-POMDP in Equation 3.8. Again, as stated in Section 2.1.2, the purposed I-POMDP allows the agents to work separately and interactively update the global state. (Note this is only an initial inclusion of the I-POMDP model. Future work should capitalize on this model, using the construct to further define the solution.)

$$I_i = \{IS_i, A, T_i, \Omega_i, O_i, R_i\} \tag{3.8}$$

The interactive state $IS_i$ depends on the state $S$ and the belief that other agents interact with that state, $\theta_j$, using $IS_i = S \times \theta_j$.

*Individual Agent State*

The state of the agents in the I-POMDP motivates a further description of the agents. Looking from the agents perspective on the domain, many local knowledge pieces are articulated. Equation 3.9 shows the pieces of the local knowledge need in this sub-model.

$$S : \upsilon(\lambda_{\upsilon t}, e_\upsilon, d_\upsilon, \eta_t, \tau_{\upsilon t}, \beta_t) \tag{3.9}$$

The $\lambda_{\upsilon t}$ defines the location and velocity vector from before, along with the detection $d_\upsilon$ and engagement $e_\upsilon$ range. It also adds $\eta_t$ as the neighborhood of agents,

$\tau_{vt}$ as the set of targets currently known by the agent, and $\beta_t$ as the behavior set. All of these aspects of the state are used in the development of the SO system.

Equation 3.10 [30] defines the set of agent actions $\Upsilon$:

$$\Upsilon_t \rightarrow I\{I_1, \ldots, I_i\} \tag{3.10}$$

thus, defining the new state from the transition of each agent which derives from the interaction of the other agents.

*Observations*

Equation 3.11 represents the observations of the domain with the same dependencies on the I-POMDPs from the transition set:

$$O : P(s') = P(s', \Phi, s)P(s) \tag{3.11}$$

The observations are taken from the set of observations $\Phi$. The observation function of each agent is defined by Equation 3.12

$$O_i : IS_i \times A \times \Omega_i \tag{3.12}$$

Where $IS_i$ is the agent state dependent on interaction, $A$ is the action and $\Omega_i$ is the individual observation. From this the observation set is defined $\Phi : O_i$.

*Reward*

Reward functions as a push back on the system to "optimize" the cooperative control structure that reacts to an abstracted state. The reward policies that address the state transitions come in the form of SO rules. The basics of this relationship are shown in Figures 2.12, and 2.13. From the possible behaviors the effects of moving through a hostile environment evolve emergent swarm behavior. The most important reward/feedback comes in the form of survivability, $\omega_j$, making it through the envi-

ronment without crashing or being eliminated. The second functions is the ability to successfully engage targets and destroy them, $\kappa$. The objective values are returned from the reward state-action pairs reflected in Equation 3.13 in reference to $t$ time.

$$\mathcal{R}(s, a, s') \rightarrow (\omega_{jt}, \kappa_t) \tag{3.13}$$

Further development of the I-POMDP mathematical model would provide predictability in the effectiveness through a detailed system verification (proof) of performance. At this point in the design process the system develops from the substates of the model but it does not require full articulation of all the mathematic structures. Since the low-level sub-state design is done from an engineering perspective, further mathematical decomposition does not directly effect the associated computational validation. For this reason the decomposition is continued from this level of the model, which is only a reasonable representation of the Real-world.

*3.1.3 Real-World vs. Simulation.* Looking at the vastness of the problem domain given the complexity in Section 2.1.1, finding policies that fully articulate the space is infeasible. For this reason a look at the constraints of the problem and how they approximated at the higher level gives insight to the true scope of the problem domain in simulation. Many constraints exist on the agents traversing the real-world domain. They include:

- Flight dynamics of the aircraft
- Physics constraints of not only the craft but munitions
- Sensors range constraints and noise
- Communications bandwidth constraints and unreliability
- Geographic incursion on flight, sensors, and communications
- Fog and Friction of battle

All of these constraints create a scenario where relying on details of a state can cause problems as they may be hidden. Upon those constraints, simulation world also places its own set of constraints:

- Computational speed limits which restrict size and accuracy
- Memory constraints
- Simulation of communications links without utilizing unstable medium backbones
- Complexity of dynamics on the agents and target restricted to known computational models
- Scenario variability constricted to expert knowledge

As a result, the system and simulation can only function with a restricted amount of validity compared to the real world. However in SO decomposition, the systems are biologically inspired. This creates a juxtaposition between the human need to thoroughly develop a state and reality of the level of states that are used by the exemplar biological agents. With SO we tend to steer away from exact state modelling and allow the system to abstract the states to the level needed for survival and attack in the given domain.

*3.1.4 Abstracted State Architecture.* The abstraction of states in highly complex domains allows Self Organized systems to thrive. The greatest example of this is the worker ant. When building an agent removes all extraneous state information and considers only at the relevant local state. There is a rock in front of the agent and it is not part of a wall, pick it up [15]. Once the ant bumps into a wall it places the rock. That is the extent for that mechanism. It finds its way to a wall via pheromone, that however is a separate mechanism. It too is simple, move along an upward gradient of pheromone. If systems can decompose the state, simple sets of rules can be formulated. This approach does require selection modes. The mode

selection only requires basic state information at that level. The selection functions similarly to the modes, the system functions efficiently on the abstracted state.

Application of this strategy comes into play during the decomposition of the problem domain and emergent structure in not only the behaviors but the control structure itself. To do this it requires the 'U'-decomposition process as shown in [53]. With this approach it is not enough to simply decompose the problem into pieces top-down, but the bottom up engineering of the supporting structure proves to be as crucial. Taking a group of simple rules and applying them to a top-down structure does not constitute SO engineering. For this reason all aspects of this system must be approached in the same way. Figure 3.1 illustrates the 'U'-decomposition technique.

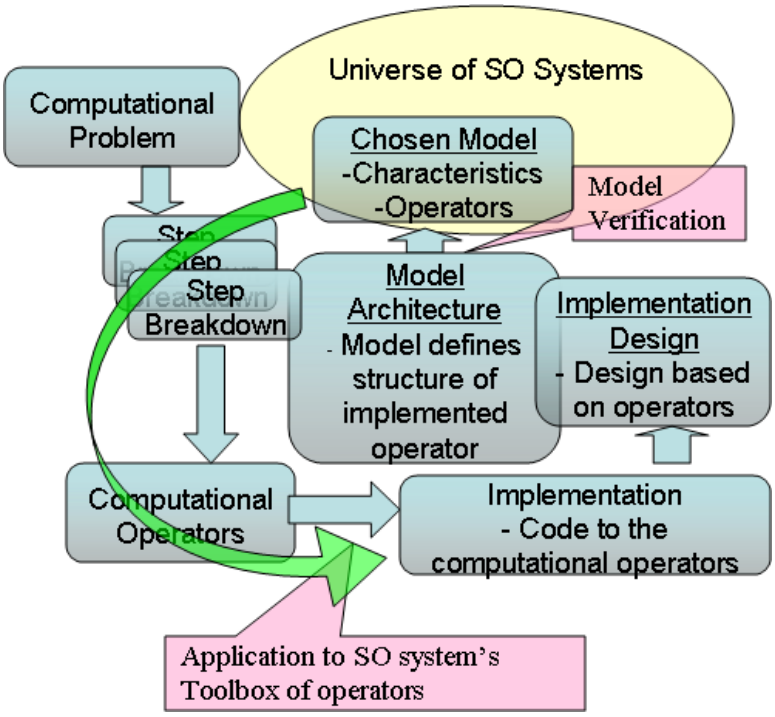


Figure 3.1: 'U'-decomposition technique

As a result of this design approach, coupled with an object oriented program, the system develops a structure that has a "entangled hierarchy" with abstracted states governing the inclusion of the rule sets. The state moves into the system and is quickly reduced through compartmentalized rule sets. As a result, the state information self organizes in such a way that only the applicable data sets are pulled

together. Generalizing pieces of the state in this way allow the behaviors to act dynamically on only the information they need.

In concert with this abstracted state the system develops dependencies that are otherwise independent SO behaviors. An entangle hierarchy forms as a result of two aspects: this abstracted state and inheritance issues between complex rules sets and simple rules. This dependency on information and skill sets of other behaviors creates synergistic effects on the global system, as seen in Figure 3.6. These two ideas must be appropriately addressed when moving into the next phase of algorithm domain definition.

## *3.2 Algorithm Domain - SO Rules*

We continue the 'U'-decomposition to develop specific SO rules, their emergent structure, and the Genetic Algorithm (GA) for search to define policies that allow successful traversal of the POMDP search space.

*3.2.1 SO.* The focused problem domain primarily addresses the interaction between the targets and the agents. This is a sub set of actions and transitions.

Action sets take two forms, movement and engagement. Movement happens in two abstracted states, when targets are detected and not detected. Equation 3.14 shows the set of movement actions.

$$\{\mu_{A_d}, \mu_{A_{nd}}\} \subset \mathcal{A}(\mu, \varepsilon) \tag{3.14}$$

Where $\mu_{A_d}$ are actions during detection of targets and $\mu_{A_{nd}}$ are actions without target detection. In the same way, Equation 3.15 shows the subset of engagement actions in this abstracted state.

$$\{\varepsilon_{A_d}, \varepsilon_{A_{nd}}\} \subset \mathcal{A}(\mu, \varepsilon) \tag{3.15}$$

The transitions add one more set. The detected target set derives to a set of reconnaissance and engage. Equation 3.16 represents the result on the transitions.

$$\mathcal{V}_{nn} \cap \mathcal{V}_{nd} \cap \mathcal{V}_{ne} \subset \mathcal{T} \tag{3.16}$$

$\mathcal{V}_{nn}$ shows the transition set without detection, $\mathcal{V}_{nd}$ dictates detection without engagement, and $\mathcal{V}_{ne}$ includes engagement. Although this seems more complex the reduction of these states to abstract sets facilitates easier transition decisions.

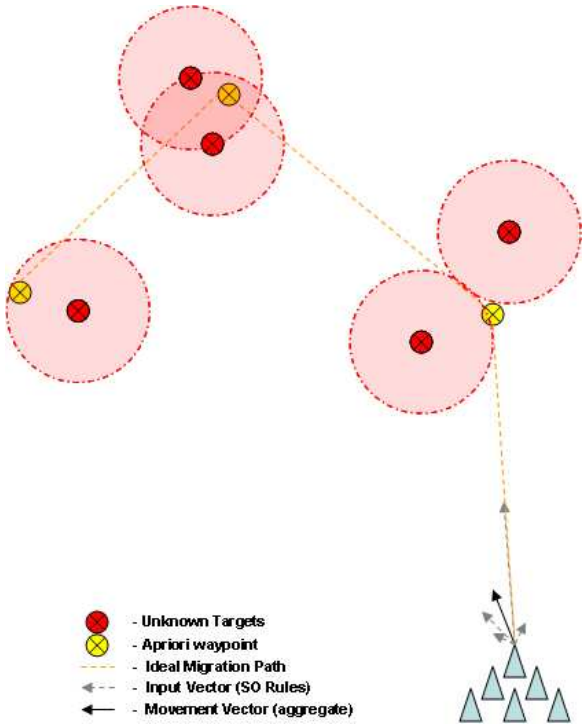


Figure 3.2: Migration patterns relative to target areas.

*Migration*

The migration transition vector set only belongs to the non-engagement abstract states $\mathcal{V}_{mn} \cap \mathcal{V}_{md}$. Figure 3.2 shows the pictorial definition of migration through waypoints. To reiterate, the waypoints in this domain are not hard waypoints, but rough estimates of the location of hostile forces.

The swarm uses the waypoints to reach the approximate target area. To allow migration to react dynamically, each agent decides which waypoint is closest and moves toward it. As a result the swarm may split if two waypoints are equidistance from the swarm. Restricting the agent to abstract state knowledge, this rule requires the agents current location and a list of waypoint locations. Equation 3.17 shows the extension of the states to include the waypoints.

$$\mathcal{S}(\upsilon, \tau, \zeta, \chi) \tag{3.17}$$

*Localized Target Engagement (Bee-Inspired Attack)*

LTE is where "the rubber meets the road" for target engagement. The intent is to move past the benefits of a simple mass attack. We focus on the interaction between the swarm, with its emergent behaviors, and the target sets. Optimizing in this environment is a function of damage vs. causalities. In order to work and train this system however more advanced scenario sets must be used to focus the efforts of the algorithms used to address this MOP, shown in Chapter V. The first task accomplishes the 'U'-Decomposition.

*Mapping to Algorithm Domain*

A subset of the problem domain is target engagement. For the purposes of this research effort, agents reach the target engagement sub-state when the system detects a target. Equation 3.18 defines the sub-state in terms of the POMDP state tuple.

$$\begin{aligned} & D(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{R}) \\ \forall \mathcal{S} \epsilon \upsilon & (\lambda_{\upsilon t}, d_{\upsilon t}, e_{\upsilon t}, \eta_t, \tau_{\upsilon t}, \beta_t) \\ & where\ \tau_{\upsilon t} \neq \emptyset \end{aligned} \tag{3.18}$$

There are three subsets of states that follow: when the agent has only detected a target, when the agent has the information to make a decision on target engagement priorities, and finally the actual target engagement.

*Mapping to SO rules* This rule belongs to a subset of transitions, the detection and engagement $\mathcal{V}_{ed} \cap \mathcal{V}_{ee}$. A subrule takes control based on the knowledge over the target area, explore, vote, attack, $\mathcal{V}_{ed_x} \cap \mathcal{V}_{ed_v} \cap \mathcal{V}_{ed_a}$. Abstracted states take over to control the define the abstract state and allow for hand off to the subrules.

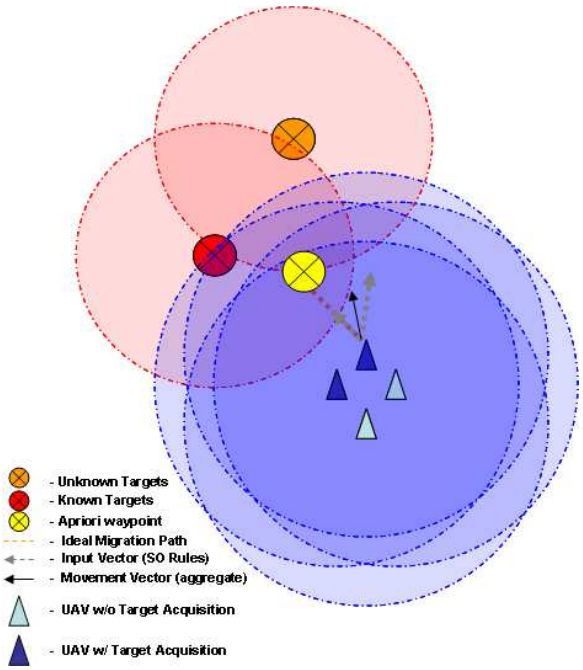


Figure 3.3: Migration patterns draw swarm to detection of target sets.

The exploration is shown in figure 3.3. First subrule calls for disengagement from migration and then local reconnaissance for other targets that could aggregate defenses. Communications allow distribution of state information as the agents move around the target. This subrule is modelled on the Bee Exploration metric where a set of bees look for a new hive location, each analyzing itself and providing the information to several other local bees who in turn explore and analyze. [71]

Figure 3.4 shows the swarm decision process during exploration. Once all of the local information is gathered each agent makes a decision on the most vulnerable

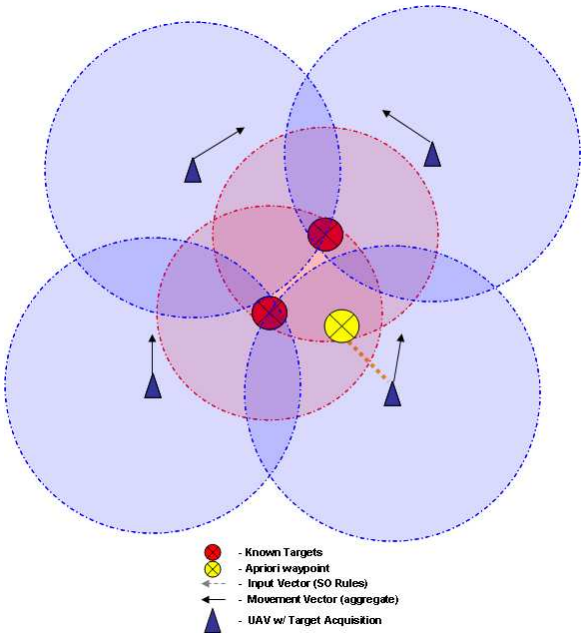


Figure 3.4: Each agent based on the abstracted state defines the strength metric that moves it towards the most vulnerable target.

target. This is based on the location and overlap. A vector draws the agent towards the weakest. This Decision process is based on the bee hive selection model in [71,79].

The coordinated attack happens when through implicit communications the threshold to engage successfully has been met. This is stems from to the 15 bee threshold that Visscher shows in [79]. Once that threshold level is meet the swarm knows to move, implicitly.

Figure 3.6 roughly shows the entangled hierarchy for the subset of rules related to target engagement. In most cases the SO behavior interaction is straightforward, several behaviors act as multipolar balancing forces to create a simply emergent behavior. This facilitates simple, emergent, state-action pair behavior.

Braitenburg [13] outlines layered computational cognition layers and develops a simple agent only capable of the most fundamental of those layers. Research into developing agents that accomplish high cognition on simple rules sets have created a plethora of control structures that complicate states and require heavy iron (powerful computers) to operate! With the SO decomposition the control structures are

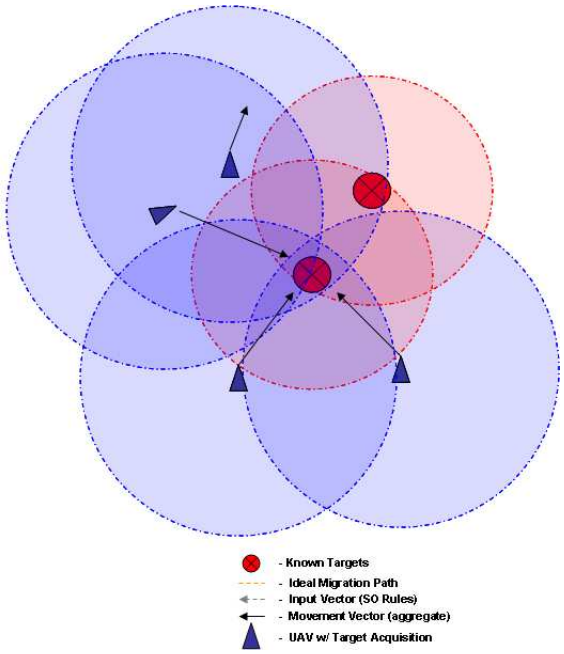


Figure 3.5: Attack is based on the number of agents in the swarm, the location of the targets and the number agents ready to engage.

developed from the behaviors. This approach focuses on finding the crucial state information and allowing the structure to form around the interplay of the behaviors and that abstracted state. This gives way to an emergent structure that is system/agent dependent.

Again, Figure 3.6 the basic behavior set at the lowest level. The resulting structure, although not complicated in this simpler case, does result in an entangled dependence on state and action set information. The result is a structure that forms with low dependency on state and high levels of re-utilization of behavior information.

These two new behavior rules provide input in to the control and movement of the UAVs in the form of a weighted vector. The weights of the vectors and a several control parameters are added to the chromosome shown in 2.2.6. Also the POMDP model description and decomposition ends here. At this level the abstract states from the decomposition of the domain description are imbedded. As a result any information needed from the model for control or structure utilizes the properties of the operators from this level.

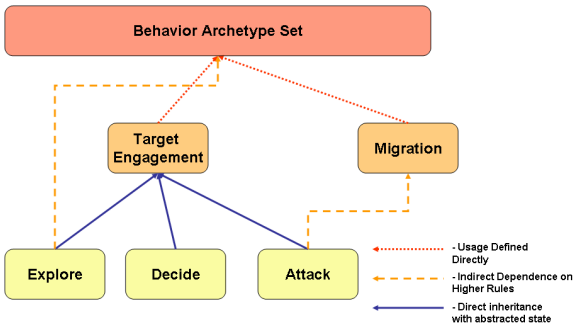


Figure 3.6: Relationships based on the direct, indirect or abstracted state translation.

### *3.3 Algorithm Domain - Self Organized Genetic Algorithms*

This section outlines the high level design of the GA that is used to optimize the control, rule and parameter weights. The implementation in MOP space and the requirement to focus further research motivates the development of this SOGA. SO decomposition in this case forms around the known structure of the GA. The simple operators of the GA are then developed using SO priciples.

In the natural world many systems develop through Self Organization(SO), emergent properties as a result of localized agent interaction sans global knowledge [15, 23, 35]. In DNA there is no global knowledge of the DNA structure from the allele's/nucleiotides perspective, however the nucleotides do interact controlled by protiens and structure does emerge. Although no proof exists in biochemistry, yet the possibility that these agents, alleles/nucleiotides, have properties similar to other SO systems exists. This is the foundational impetus for Self Organized Genetic Algorithms (SOGA)!

With SOGA whether or not SO happens in DNA matters not, the point is bringing a tool in that has worked in other computational venues. Why SO? There are three benefits using SO decomposition in computational problems: ease of implementation, lowered computations, and dynamic adaptation. Using SO implies finding some set of behaviors from which a desirable structure emerges, if done properly these behaviors should be easily coded. Lower computational cost stems all computations

executing localized at the agent level and interaction and communications are also very localized. Finally, dynamic response happens as emergent behaviors do not restrict the system to a script but instead are capable of quick response to unpredictable stimulus through those rules.

What we are looking for in terms of an SOGA is the response to problems in a dynamic nature enabling more versatility and universality. To do this we remove some of the restrictions, problem specific constraint, niche operators, and parameter tuning commonly used [20, 82]. We attempt to finally "bag that white rabbit" by allowing the population to tell the algorithm what it needs. With higher population commonality the algorithm senses building blocks/genes with successful values and allows them to thrive while still varying aspects with lower known probabilities. When the problem space is first being explored or in a state of high exploration the algorithm responds by continued exploration. To do this we add an operator to sense the current genotype space, act upon that information, and then explore the highly varied world of recombination operators to find the appropriate application of that information. Recombination has been the studied extensively spawning many different approaches to satisfy different problem constraints.

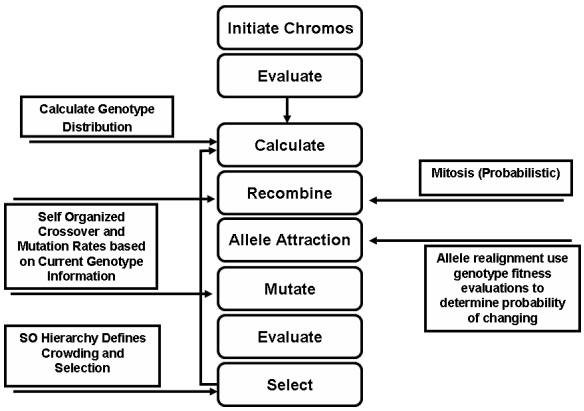


Figure 3.7: Self Organized Genetic Algorithm SOGA

*3.3.1 Genetic Distribution Collection.* Figure 3.7 shows the required additional operators and algorithm flow. The Calculate step determines the Genetic

Distribution Collection(GDC) in the population to help facilitate the sensing of the exploit versus explore aspects of the rest of the algorithm. Information from the GDC facilitates evolution rate updates. The recombination step is then modified to include knowledge gathered from the GDC. The Correcting Allele Attraction (CAA) operator also utilizes this information through probabilistic interpolation of the space. And the selection operator utilizes a new crowding distance operator.

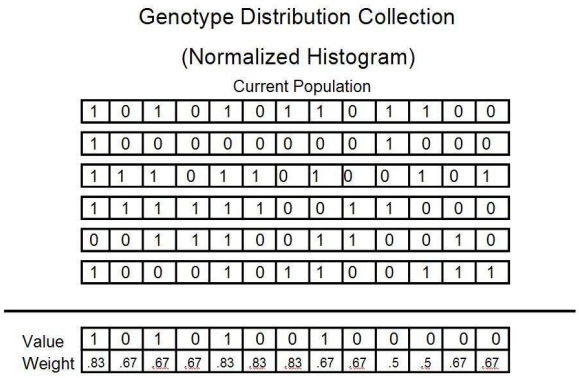


Figure 3.8: Genetic Distribution Collection GDC (Normalized Histogram)

Figure 3.8 shows the simple function of calculating the Genotype Distribution Collection. It simply utilizes a data structure, $\Gamma$, similar to a chromosome to store the highest likely value for an allele and the normalized histogram weight of that value. This histogram gives a reading to the system of overall entropy defined by Equation 3.19

$$H = Kln(\frac{1}{p})\ \epsilon p = \sum w_j \tag{3.19}$$

and also the location of unstable alleles which is defined by equation 3.20:

$$S \rightarrow w_j \tag{3.20}$$

$w_j$ is the weight of the allele defined by the number of corresponding allele values in the population over the total size of the population.

*3.3.2 Crossover and Mutation.* Initial application of the insight gained from the GDC allows the formation of the mutation and crossover rates. By watching the changes in the GDC from the last generation the system can determine the amount of entropy in the population. With lower entropy the system continues to maintain high levels of crossover and mutation. Of course with the higher entropy the system has moved into the exploitation phase of the algorithm and does not require as much variance in the chromosomes. Equations 3.21 and 3.22 represents the updating function for the rates of mutation and crossover.

$$c = \frac{\sum_{i=0}^{chromo_{len}} (\Gamma_i(t-1) - \Gamma_i(t))}{\Gamma_{size}} \tag{3.21}$$

$$m = \frac{\sum_{i=0}^{chromo_{len}} (\Gamma_i(t-1) - \Gamma_i(t))}{\Gamma_{size}} * max((\Gamma_i(t-1) - \Gamma_i(t)) \tag{3.22}$$

*3.3.3 Mitosis.* The second operator change comes in the recombination step. The system recognizes when good building blocks exist and attempts to perpetuate their existence. As shown in Figure 3.9 the system analyzes the $\Gamma$ level to determine the strength of each allele in the crossover section. From that, the system probabilistically chooses between mitosis, which facilitates exploitation, and meiosis, which enables exploration, recombination based on a threshold $\varpi$. If the normalized summation of the alleles in the crossover section is above the threshold it chooses mitosis on the higher gene based on that probability. The result is shown in the third pair of chromosomes in Figure 3.9.

*3.3.4 Correcting Allele Attraction.* The CAA utilizes that same GDC information and exploits it to establish linkages between pairs or subset of disjoint alleles. As in the modified crossover, this operator allows the system to focus on high exploitation when the population is diverse and solution likely unknown while exploiting known sets, building blocks, or linkages. Here $\Gamma$ analyzes every allele, and does a replacement based upon probability from equation CAA.

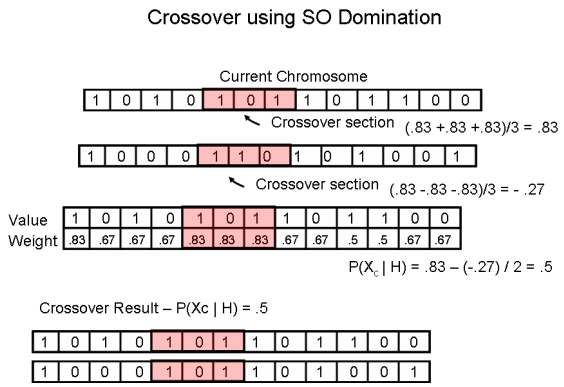


Figure 3.9: Crossover using GDC Figure 3.8 to determine the probability of meiosis and mitosis.

$$P_{change}(x|\gamma) = \Gamma_{w_i} * (W_c - W_n) \tag{3.23}$$

Here both $W_c$ and $W_n$ are the normalized summation of correct and incorrect mappings, respectively, between $\Gamma_v$ and $\vec{a}_t$.

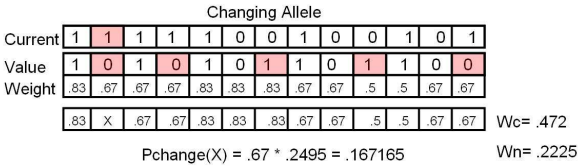


Figure 3.10: Changing the allele in red on the top chromosome through the weighting of the the GDC figure 3.8 for this generation. The result of the GDC for correct predictors ($W_c$) and non-correct predictors ($W_n$) is normalized and added. The probability of change is the product of incorrect prediction of the given allele and that difference.

*3.3.5 Relation to MOMGA.* In the [78] the idea of Multi-Objective Messy Genetic Algorithms (MOMGA) is introduced. With a MOMGA the algorithm searches for building blocks that result in higher levels of fitness. These building blocks are built off the schema. Understanding how to analyze the chromosome for building blocks and which building blocks are good is difficult. We allow the selection operator to determine what is good. Through the schema theorem we know that good building blocks perpetuate themselves over time. Looking at the make up of the

population shows which pieces are increasing in occurrence, which given the selective nature of GAs, implies the effectiveness of that building block. $\Gamma$ provides all of this information.

Looking at the entire chromosome through $\Gamma$ allows the system to develop connection throughout. This allows building blocks that are geographically separated. Through the Markov assumption the system can adapt the building blocks to the current populations. These dynamic capabilities allow for more effective utilization of the building blocks.

*3.3.6 Selection.* Selection in the natural world stems from environmental pressures. In order to continue to place pressure on the population, the algorithm uses a form of elitism. SOGA utilizes the same *fastnondominatedsort* as NSGAII [26]. However, the crowding operator uses a self organized ranking structure, illustrated in Figure 3.11. First the neighborhood gets defined dynamically by the size of the current population space in all directions of every objective. Then the individuals that qualify as neighbors use a SO ranking structure similar to that outlined in [76]. The remaining positions in the child population are filled based on the ranking structure. Initially this generates a distributed set of the less fit individuals in a rank. When the higher ranks become more crowded the algorithm pushes the individuals towards the less explored reaches of the front.

With the selection operator applying pressure, the crossover and mutation exploring and the CAA acting as a self correcting gyroscope. The system finds a balance in exploitation and exploration. With the inclusion of both parent and child in the selection population, the algorithm allows good material to be carried not only by the genetic material but through experience of the old chromosomes as well.

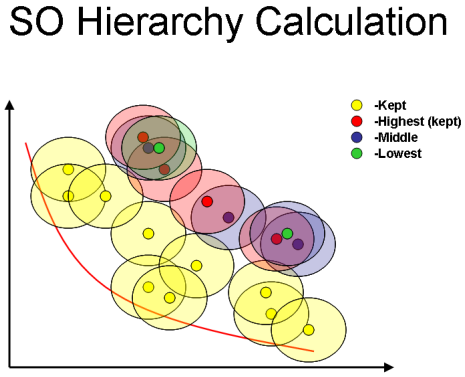


Figure 3.11: SO selection operator example. The inner circles are the individuals in the population. the outer circles represent the neighborhood. The yellow individuals are kept because of their rank, the red individuals have the highest levels in the SO hierarchy and are also kept.

*3.3.7 Bäck Notation.* Bäck's [7] notation serves as the foundation for the derivation of the rest of the MOEA. This construct allows algorithm development to be fully expanded and thoroughly vetted.

$$EA = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda) \tag{3.24}$$

Very little of the construct changes for SOGA. Just a few of the operators must be formally described.

$$\omega_{\Lambda} \epsilon \Omega \ni \Lambda_i \leftarrow \mathcal{GDC} \tag{3.25}$$

$\Gamma_i$ defines the operator for collecting the histogram information from each allele in each chromosome.

$$\omega_{\Theta_c} : I_1^{\lambda} + I_2^{\Lambda} \rightarrow I_{12}^{\mu} | I_{11}^{\mu} \tag{3.26}$$

Equation 3.26 shows how the crossover can take the form of meiosis $I_{11}^{\mu}$ or mitosis $I_{12}^{\mu}$.

$$\omega_{\Theta_{cm}} : C_{rate} \& M_{rate} \leftarrow \mathcal{GDC} \tag{3.27}$$

Equation 3.27 defines the adaptive rates of crossover and mutation as a function of the GDC.

$$\omega_{\Theta_{caa}} : I^{\lambda}_{CAA} \leftarrow \mathcal{GDC} \tag{3.28}$$

Equation 3.28 also defines the CAA as a function of the GDC. And finally, the selection operate shown in equation 3.29 results from the SO hierarchy.

$$\omega_{\Theta} \epsilon \Omega \ni S_i \rightarrow (\mathcal{F}_1, \mathcal{H}) \rightarrow (i_{rank}, R_{SO}) \tag{3.29}$$

*Fitness Function*

In order to define the fitness function, we must first analyze the objective functions. Attack UAV swarm focuses on destroying targets. Equation 3.30 defines values based on successful engagement.

$$D_t = \tau_{destroyed} * 100 + \tau_{destruction} * 10 \tag{3.30}$$

The second fitness function is the casualty rate, defined in Equation 3.31

$$C_t = \nu_{damage} * 10 \tag{3.31}$$

The damage received is multiplied by ten to keep it in scalar concert with the damage inflicted. Complete destruction of an agent results in a score of 100.

*Generate Population*

In initialization each allele (bit) of the chromosomes gets chosen using a random number generator. After five bits are set the gene gets placed on the chromosome.

The process repeats for the length of the chromosome. If seed chromosomes exists, they are added to the population otherwise every chromosome gets created through this process.

$$I = A_x \times A_s where A_i \Rightarrow \omega_{\theta_i} = rand(bit) * length\_gene \tag{3.32}$$

*Evaluation*

To evaluate the fitness of this chromosome, a simulation of the given scenario runs and returns the score. The damage inflicted and casualty information forms from the statistics collected in each simulation. The result of the fitness function calculation on this data determines the overall fitness.

*Crossover and Mutation*

Normal crossover and mutation rates and controls apply. The difference in this implementation comes in what gets modified. In both instances an entire BA gets modified. The complex form of the chromosome is particular to this problem domain. This forces the evolutionary operators to specifically address the points at which changes are made. In mutation, changes in alleles happens in a single BA with both the control section and behavior section of the chromosome. In this design each part of each BA mutates.

$$\omega_{\Theta_{mc}} : I^{\lambda} \rightarrow I^{\mu} \ni m(\mathcal{BA})|c(\mathcal{BA}) \tag{3.33}$$

*Selection*

The algorithm uses elitist for generational selection. With the space as diverse as it is and the population and reproduction operators facilitating high levels of exploration, elitist approach allows the algorithm to exploit the good genes. [59]

$$s_{\Theta_s} : (I^{\mu+\lambda} \rightarrow I^{\mu}) \tag{3.34}$$

Allowing both parents $I^{\lambda}$ and children $I^{\mu}$ to continue to the next generation.

The inclusion of a system entropy gage allows for appropriate application of evolutionary pressures. The mutation and crossover methods address the issues inherent to the structure of the control structure for these chromosomes. The Correcting Allele Attraction attempts to find geographically separated building blocks. Finally a SO hierarchal operator assists in the selection. This SOGA approach makes it possible to fire and forget about the GA optimization aspect of this research tool.

## *3.4 Summary*

This chapter form the top-down decomposition of the problem in the form of the POMDP model. The problem domain focused on the target engagement aspects of the domain. It then decomposed the problem and GA into small implementable subsets. This laid the ground work for the implementation of each of the pieces. The next chapter starts from the implementation and discusses the back half of the 'U'-decomposition.

# IV. SO Implementation and Structure Design

This chapter describes the data structures and algorithms of the implementation and low-level design of the system. The definition of the derived SO rules and their emergent structure are shown. Specifically, they focus on the low level design and implementation for two new SO rules, using the new SO decomposition techniques. Continuing the bottom up aspect of the SO 'U' decomposition from Chapter III, that is followed by the description of the control new structure. The outline for the algorithm implementation of SOGA and its emergent structure derives the implementation environment. Finally the abstracted emergent structure is shown completing 'U'-decomposition in both cases.

## *4.1 SO Rules for Target Engagement*

At the intersection of the basic problem domain pieces with SO rule instantiation, the real intricacies of the SO rule interaction and the resulting emergent behavior start to form from the 'U' Decomposition. As such, both pieces are described. The resulting macro-level structure can not be explicitly extended, as the formation of the synergistic swarm is probabilistic and dependent on the application and scenario definition. The intent of an SO system is to allow the agents to use simple rules that build up synergistic behaviors that adapt to dynamic situations. For this reason the precise response to a particular situation or the expanse of the domain of responses can only be described abstractly.

*4.1.1 Data Structure.* The data structure for all of the behavior rules varies. Table 4.1 shows the basic state utilized by the migration, Section 3.2.1, and Bee-Inspired Attack, Section3.2.1, behaviors.

Notice the system does not store the entire scenario space. In all cases, however, the system works with some subset or abstracted version of the UAV state. Key to this data structure is the position, direction, bearing and force vectors, behavior matrix (BAs), neighborhood agents, known target, waypoints and obstacle lists, target marks,

Table 4.1: Abstract State Requirements

| Migration | Bee Attack |
|---|---|
| Current Position | Current Position |
| Way-Point Set | Target list and Positions |
| Way-Points Achieved | Recon Points |
| | Chosen Target |
| | Neighbor votes |

sensor, munitions, and communication ranges. Several methods beside nextposition() and newdirection() exist as well: attack, update, noisy movement, and three sensor weight calculators. In most cases only a very small subset of the UAV state is needed in any behavior.

**Algorithm 1** Migration Algorithm

```
1: procedure MIGRATION
2:     for i: 1 to length χ do
3:         d_νχ = χ_l − ν_l
4:         if d_νχ < d_best then
5:             d_best = d_νχ
6:         end if
7:     end for
8:     getTransitionVector χ_best, ν
9: end procedure
```

*4.1.2 Migration.* Extending from Section 3.2.1 the migration rule implements the behavior to take any set of waypoints and move towards them. It acts as another force on each agent that gets aggregated into the movement vector. Because behavior is agent based, the situation could arise where a swarm separates dynamically. In concert with the SO decomposition and the need for robustness, the low level design must remain generic.

For the purposes of testing and developing the swarm capabilities inside this simulation environment, migration is focused on simple tasks. Waypoints are set up in the regions of the target areas. A simple algorithm determines at initialization of the simulation, where clusters of targets lie. Waypoints are created at the center of

mass of those targets. Then a probabilistic variance is applied producing a behavior that does not draw the agents directly to the target each time.

Algorithm 1 in 4.1.2 matches what is illustrated in Figure 3.2. For any given agent the closet waypoint defines the output vector. The normalized vector integrates in with the other behaviors.

*4.1.3 Bee Attack.* From Section 3.2.1 three aspects or phases of localized attack must be addressed. Once the agent acquires a target through communications with another agent or detection from its own sensors, it immediately changes to reconnaissance mode. After the agent enters reconnaissance mode it constantly checks if it has enough the information to analyze the target area. (Note: the system does not attempt to analyze the target each iteration). During the analysis phase the set of known targets are chosen based on strength and yield. Finally the agents line up to engage the targets; agents communicate their chosen target to its neighbors and only proceed when the conditions are favorable. As each of these pieces are formed, the control structure for this more difficult sequential behavior emerges. Figure 4.1 shows the Bee attack state diagram.

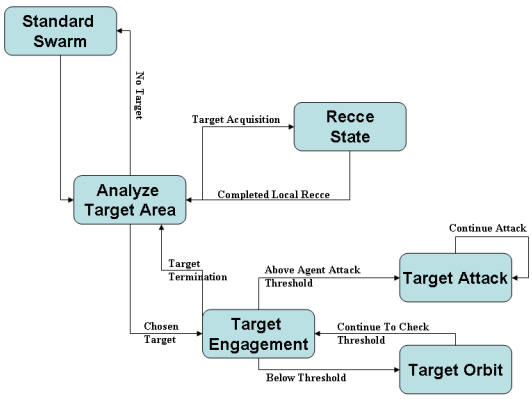


Figure 4.1: The state diagram for the Bee Attack showing the data flow and state changes.

*Target Acquisition* In target acquisition the assumption is that sensors, communication lines, and target recognition capablities are all functioning properly. The agent has a running set of all known targets and updates the states of the targets

through sensor readings and communication with neighbor agents. The list of targets includes local targets and those found by agents in other parts of the space and communicated through the network.

*Local Target Area Recon* The intent of this behavior revolves around getting enough information of the local area, around the target, to make an educated decision on target strike sets. This is closely modelled after the bee nest discovery in [71]. Dozens of bees are sent out to accomplish independent reconnaissance of nesting locations. Possible hive locations are judged on size, shape and location, taking into account weather, food source, and protection. Overtime interagent communication attracts bee to better hives. Once a threshold is met, the swarm moves into the new hive location. Another alternative to this implementation is the use of ant pheromone trails. Implementation of this would only lead agents to a target but not show any discretion in terms of target classification. A similar insect the Wasp has as attack focuses on single agent attack and retrieval, not applicable to this domain.

The algorithm focuses on the simple task of exploring the area. The agents keep no information, extraneous to the viability of the target, about the area being explored. This amounts to only adjacent target locations. With independent agents taking independent surveys, the system as a whole utilizes statistics to accurately survey the target area.

**Algorithm 2** Target Recon Algorithm

```
1: procedure RecceTargetArea
2:     if (targetList = null and recceFlag = null) then
3:         SaveState(position, heading)
4:         throwRecceFlag
5:     end if
6:     if !TargetList.contains(currentTarget) then
7:         TargetList.add(currentTarget)
8:     end if
9:     orbit(closestTarget)
10: end procedure
```

Algorithm 2 in 4.1.3 represents the bee inspired reconnaissance rule. The agent uses simple triggers to determine if it requires reconnaissance. Saving the state in line 3 allows each agent to determine if it has completed the loop around the target area and returned to its original spot. Observe that the orbit function derives from the other target orbit behavior, Figure 4.2 shows two UAVs orbit in reconnaissance mode. This minimizes computation and creates interdependence on the same sensor weight data.

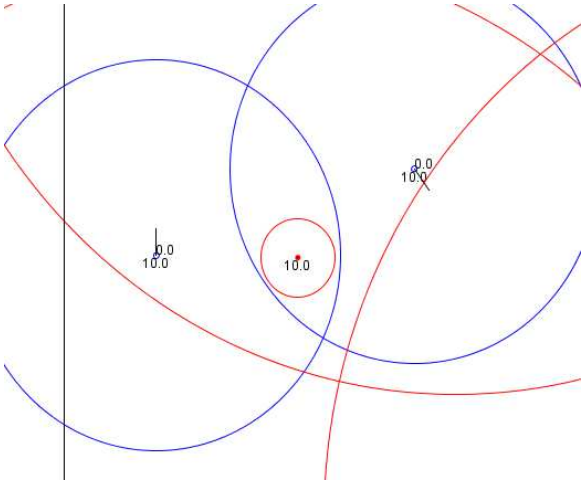


Figure 4.2: Two UAVs in local reconnaissance mode. The lines extending from the blue agents are instantaneous heading vectors.

*Target Analysis* To reiterate, the target analysis happens in a self organized manner, maintaining agent independence. Bee's give us no inspiration on this aspect of the attack. Instead we draw from Sun Tzu [77]:

> Military tactics are like unto water; for water in its natural course runs away from high places and hastens downwards.
>
> So in war, the way is to avoid what is strong and to strike at what is weak. [Like water, taking the line of least resistance.]
>
> Water shapes its course according to the nature of the ground over which it flows; the soldier works out his victory in relation to the foe whom he is facing.
>
> Therefore, just as water retains no constant shape, so in warfare there are no constant conditions.

To summarize this Sun Tzu passage, we need to apply the greatest amount of our force at the enemies weakest point. In AFDD-1 [39] one of the key principles of war is mass, which focuses on applying the appropriate amount of force to key objectives. Judging that proves relatively easy. With the gained understanding of the target area, the next step of evaluation focuses on the known target threats. The system needs to work on simplified states. The only aspect of state being evaluated is the position and engagement ring of known targets, shown in equation 4.1.

$$(\lambda_\tau, *, e_\tau, *, *) \subset \tau \tag{4.1}$$

This equation defines the subset of targets used during the target area analysis. Notice the substates of the targets are only dependent on the location $\lambda$ and engagement ring $e$. Again the system focuses on the minimum amount of state.

**Algorithm 3** Target Analysis Algorithm

```
1: procedure ANALYZETARGETAREA
2:     if recceFlag then
3:         if recceComplete then
4:             resetRecceFlag
5:             analyzeFoundTargets
6:             StoreVotingTarget
7:         end if
8:     end if
9: end procedure
```

Algorithm 4 in 4.1.3 shows the basics of the target analysis. Note only when in reconnaissance mode does the system analyze the target in line 4. Algorithm 4.1.3 shows how the agent chooses the weakest target. The system gathers each target's aggregated defensive effectiveness in the set. Then it finds the subset of targets with the weakest defensive posture. If there is more than one weakest target the closest target to that agents is chosen.

*Target Engagement* After surveying the area and voting on the target, the swarm must carefully coordinate the attack. In order to do this each agent determines if

**Algorithm 4** Target Vote Algorithm

```
1: procedure ANALYZETARGETSET
2:     for all known targets do
3:         DetermineOverlappingEngagementRings
4:     end for
5:     determineTargetsWithLeastOverlap
6:     ChooseClosestTarget
7: end procedure
```

there are enough other agents to attack. It is possible to do this through implicit communication by observing the number of vehicles localized around the target, or through explicit communications, with a broadcast of the vote. If a quorum is reached the agents attack.

In this design the agents can poll the vote from their neighborhood. Given the constraints of the simulation this proved to be the best option. To optimize the threshold of the number of agents needed to attack, that value is also included in the chromosome for optimization. When optimizing this threshold value the range is constrained by equation 4.2.

$$\alpha_\tau = ((\alpha_c + \alpha_{ave})/\gamma_r) * \nu_{n_{size}} \tag{4.2}$$

The attack threshold $\alpha_\tau$ is defined by the current parameter value $\alpha_c$, parameter value average $\alpha_{max}$ (making a positive value), the genotype maximum, and the number of agents in the local sub-swarm. This allowed two things, scalability and versatility. As the sizes of the simulations change the agents are not stuck to an abstract threshold. Also if the sub swarms change in make up, the system can respond with a modified threshold.

The actual attack polling takes from the same state information that is used by the migration rule. A waypoint amounts to the actual target position. This becomes the abstracted state during the target to agent interaction.

**Algorithm 5** Target Engagement Algorithm

```
1: procedure MOVETOWARDSTARGET
2:     findVectorToTarget()
3:     getVotingNeighborhood()
4:     if distToTarget < engagementRange | numberVotes > CP1 then
5:         returnVector = towardsWaypoint()
6:     else
7:         returnVector = targetOrbit()
8:     end if
9: end procedure
```

Algorithm 5 in 4.1.3 shows the process of engaging a target after an agent has voted. In line four the algorithm allows the agent to engage the target if it is already engaged or the number of votes meets the threshold. This is important because the number of voting agents during attack could decrease, and the response without the attacking check would push the agents back out to orbit.

This rule draws from two other rules. The *towardsWaypoint()* in line 5 uses the target as the waypoint. In the same way if the agent is waiting for the proper number of votes it moves into the target orbit rule defined by Lau [47].

That the state has been decomposed and translated into rules, in accordance with the 'U'-decomposition approach to SO, emergent structure in the system appears. Figure 3.6 in Section 3.2.1 shows the entangled hierarchy of state and rules structure. The interdependence and re-utilization allows for synergistic effects, a reduced abstract state, and minimized computational needs.

*4.1.4 Abstracted States and Emergent Entangle Hierarchies.* Using this decomposition technique, however, does not give us a linear architecture. Instead as a result we see a generic structure shown in Figure 4.3. The hierarchy of control emerges from the entangled hierarchy of the state relations at the simulation, swarm and rule/behaviors level. Pulling together SO in biological agents appear much the same way, the reaction to any particular situation does not have a definitive hierarchy

but results from the agents ability to focus on the proper level of abstracted state given a situation.

*4.1.5 Emergent Control Structure.* The system needs a control structure that mediates the set of Behavior Archetypes. In order to address the complex entangled hierarchies introduced by the emergent structure in SO and dynamic nature of the environment the design of a control structure is particularly crucial. The required characteristics include the ability to handle dynamic domains, search the rugged solutions spaces created by the entangled structures, and develop multidimensional partitioning shapes.

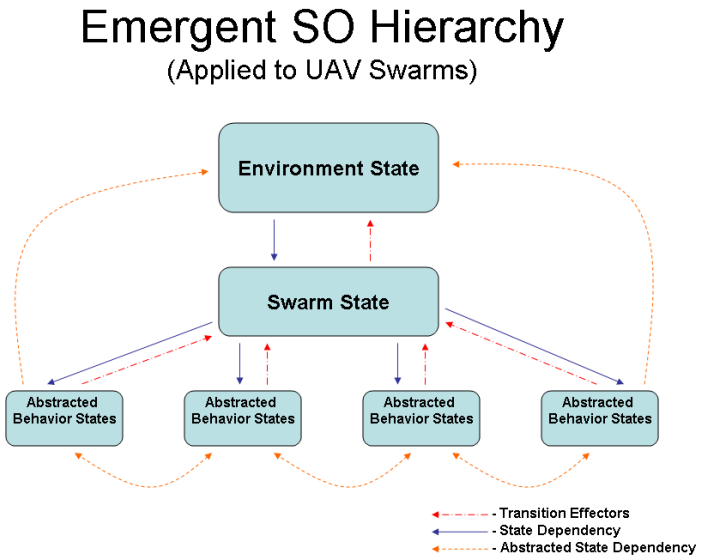


Figure 4.3: Relationships based on the direct, indirect or abstracted state translation for SO.

To address these issues, the system allows an emergent structure to form with the basic form shown in Figure 4.3. Figure 4.3 illustrates that the information and algorithmic dependencies are not strictly linear but develop based on minimization of computational and informational complex between related states. The connection of abstracted UAV state to the environment state presents an entrance point for control of the UAV's BA . Following along the same lines as earlier approaches (Sections 4.1.3 and 3.2.1) the abstracted state must be formed in such a way that systems only take

relevant information. Of course this is only a subset of the information in that state, that which is perceivable by the agent.

The previous work [59] used a controller that was based in Neural Networks (NN) . Choosing the BA in this setting, presents a problem because the space is extremely dynamic. In order to be effective NN need extensive and robust sample sets [68]. This is not always available in a probabilistic environment. In response to this phenomena, a control approach that chooses between BAs that can handle unknown states is crucial. For the solution, we borrow from the Evolutionary Computation Arena. Differential Evolution (DE) attempts to mitigate the problem inherent in a dynamic search space as articulated in Sections 2.1.1 and III, [1, 2].

Using a DE-type approach requires definition of both the data structure and the algorithm. Here the data structure comes from the abstracted state, represented by the sensor inputs. In this implementation the system uses: UAV density, Waypoint and Target proximities. Each one of these state descriptive readings corresponds to a point for each BA. They also have a weighting associated, according to their influence strength. Figure 4.5 shows the inputs readings with the corresponding data structure. The first set of numbers in the BA is the point of origin and the second is the weighting influence.

| Inputs | BA1 | BA2 | BA3 |
|---|---|---|---|
| -9 | -13 | -3 | 7 |
| -3 | 3 | 3 | 7 |
| 10 | 7 | 1 | -11 |
| | 6 | 1 | 7 |
| | 6 | 1 | 4 |
| | 3 | 2 | 5 |
| | ... | ... | ... |

Figure 4.4: Shows the sensor reading and the corresponding BA origin points and weight vectors.

This allows that system to respond to the state space. In essence it creates a sphere of influence, in the abstracted state space, on which a given BA works. Figure 4.5 illustrates this for two inputs.

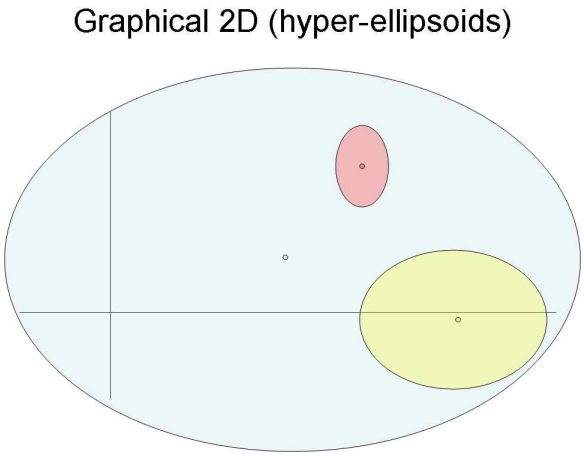


Figure 4.5: Three BA origin points and their influence hyper-ellipsoids

The algorithm itself is quite simple. The sensor inputs are matched against each of the BAs points of origin. In doing so the fitness of each BA is measured by the weighted Euclidean distance shown in Equation 4.3.

$$F = \sqrt{\sum \frac{I_k^2 - BA_k^2}{BAw_k}} \tag{4.3}$$

The square root, of the sum, of the input sensor values squared, $I_k$, minus the BA optimized values squared, $BA_k$, all divide by the corresponding weighting value $BAw_k$ gives the fitness of that BA corresponding to that given abstracted state. Figure 4.6 shows an example of the calculations.

*DE GA Operator* In keeping with the intent of DE, the GA includes a DE type operator [28]. The operator includes crossover of the control vectors in a given solution with that of a control structure of a chromosome on the non-dominated front. In this way the control vectors move towards known better solutions.

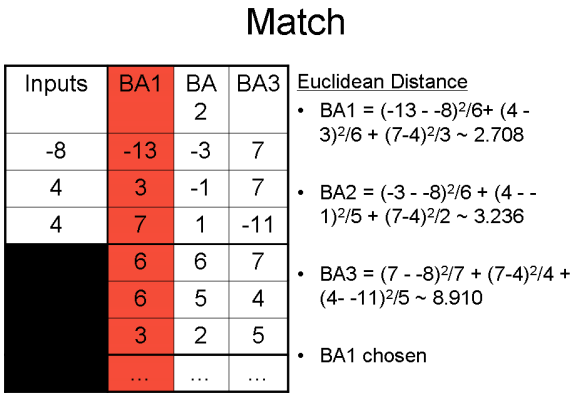


Figure 4.6: Example of the DE Matrix Matching using weighted Euclidean distance.

## *4.2 SO Genetic Algorithm*

At the implementation layer of the GA, from Section 3.3, the SO rules structure and the optimization tool are independent with the only commonality in the data structure. The chromosome serves as the interface for any control parameters that need optimization. The specific definition of the algorithm and implementation details of SOGA also utilize the 'U'-decomposition. Structure is given by the formal definition of a GA by Bäck's formal GA structure, which provides a solid stepping off point for further development. However the interplay between the operators spawns an emergent structure all of its own, shown in Figure 4.7.

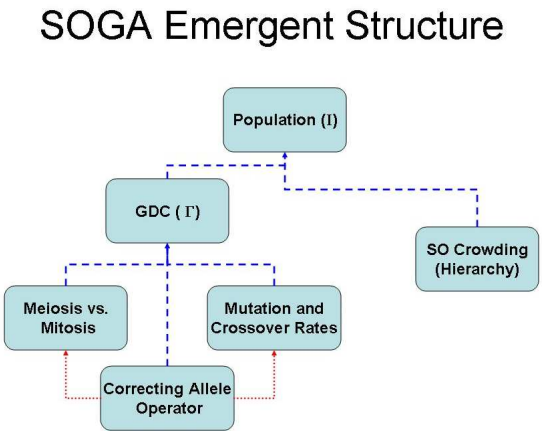


Figure 4.7: Relationships based on the direct, indirect or abstracted state translation for SOGA. Blue dashed lines show direct state abstraction. Red dotted lines show implicit states derivation.

Many of the operators draw state information from the GDC. The GDC and crowding operator, which choose representatives from heavy population areas, use an abstracted population state. Finally the Correcting Allele operator derives state from the other two mutation operators.

*4.2.1 Data Structure.* The GA works on the data structure designed by [59] and slightly modified for the new controller shown in section 4.1.5. In Low-Level design each allele is a bit, 5 bits from a gene and the number of genes is dependent on the number of rules, BAs, and other weighting parameters. Modification of the chromosome only happens at the bit level. Grey encoding is used to translate the alleles to the usable genes, as integers. This minimizes the change to the phenotype as all mutation and crossover occur at the genotype, bit level.

*4.2.2 Standard Algorithm Format.* As a result of the changes shown in 3.3, Algorithm 6 extends the generic GA Algorithm from [7]. This includes the introduction of three new operators: the $\Gamma$ in GDC, modified recombination and CAA steps, along with the self adapting mutation rates.

**Algorithm 6** SOGA

**procedure** GA
  $t := 0$;
  *generate* $P(0) := \{\vec{a}_1(0), ..., \vec{a}_\mu(0)\} \in I^\mu$;
  *evaluate* $P(0) : \Phi(\vec{a}_1(0)), ..., \Phi(\vec{a}_\mu(0))$;
  **while** $(\iota(P(t) \neq true)$ **do**
    **for** i: 1 to length $\vec{a}$ **do**
      *GDC*: $\Gamma = \sum_j \vec{a_{i_j}}/j$
    **end for**
    *recombine*: $P'(t) := r\Theta_r\Gamma_r(P(t))$;
    *CAA*: $P''(t) := r\Gamma_r(P'(t))$;
    *mutate*: $P'''(t) := r\Theta_r(P''(t))$;
    *evaluate*: $P'''(t) : \Phi(\vec{a}'''_1(t)), ..., \Phi(\vec{a}'''_\mu(t))$;
    *select*: $P(t+1) = s\Theta_s(P''(t) \cup Q)$;
    $t := t + 1$;
  **end while**
**end procedure**

The formal version of the SO GA algorithm shown in Algorithm 6 translate to the more readable version shown in Algorithm 7. Notice Algorithms 8 and 9 further articulate the *nonDominatedSortSelection* operator shown in line 15.

**Algorithm 7** SOGA Simplified

```
1:  procedure GA
2:      initializeGenerations
3:      generateRandomPopulation
4:      evaluatePopulation
5:      while !numberOfGeneration do
6:          gatherPopulationEntropyInfo
7:          if mitosisProb then
8:              mitosisCrossover
9:          else
10:             meiosisCrossover
11:         end if
12:         mutate: P'''(t) := rΘ_r(P''(t));
13:         Correcting Allele Operator
14:         evaluatePopulation
15:         nonDominatedSortSelection
16:         selfAdaptiveRateUpdate
17:         nextGeneration
18:     end while
19: end procedure
```

Algorithm 8 is the selection operator algorithm. The selection operator uses the *fastNonDominatedSort* from NSGA-II [25] to define the fronts of the population. This is used, as opposed to other MOP sorting functions, because of its speed and simplicity. In order to reduce parameter tuning and allow the system to self adapt it uses the new crowding operator, illustrated in Algorithm 9.

To design the system for self-adapt in the population centers, there are several preparatory steps. From the population size and span of the population (height and width distance between the outlining individuals) the system determines the neighborhoods size. Each individual then uses its neighborhood size to determine its neighbors. The emergent hierarchy is forced through "individual battles" with each neighbor. The strength of each individual is determined by the number of dominating fitness functions and its current rank. The winner of the battle is determined

**Algorithm 8** SO Selection Operator

```
1: procedure SELECTION OPERATOR
2:     while newpop_size + rank_size < selection_size do
3:         Newpop_add(rank(i))
4:         i++;
5:     end while
6:     sorted_rank = SO_CrowdingOperator(rank(i))
7:     while newpop_size < selection_size do
8:         Newpop_add(sorted_rank(i).(j))
9:         j++;
10:        if j > rank(i)_size then
11:            j = 0; i++;
12:            sorted_rank = SO_CrowdingOperator(rank(i))
13:        end if
14:    end while
15: end procedure
```

probabilistically and then given an upgrade in rank. Overtime the rankings produce a hierarchy. That hierarchy is used to determine the representative sub population.

**Algorithm 9** SO crowding Operator

```
1: procedure SELECTION OPERATOR
2:     for ind: individuals do
3:         t := 0;
4:         Set_Neighborhood[] → Euclidean(pop_height x pop_width)/pop_size
5:         rank[] = 1
6:         for i: 1 to Number_Neighbors do
7:             count = 0
8:             for j: 1 to Number_Objectives do
9:                 if current[1] > neighbor_fitness[i] then
10:                    count++;
11:                end if
12:            end for
13:            check = 1/(1 + e^(count+(rank_cur − rank_i))
14:            if (check > rank) then
15:                rank++;
16:            end if
17:        end for
18:    end for
19:    rank_new = rank/numFights
20: end procedure
```

*4.2.3 Application to SO UAV Swarms.* As a reiteration, the SOGA intends to optimize the control structures for the SO rule sets. With this implementation the focus of all of the testing and evaluation can be focused on the development of rules sets and the associated entangled hierarchy.

*Chromosomes*

As a continuation of the previous work [59], rules sets are mapped to the chromosomes in a similar fashion. Specific to the migration rule only the relative weighting is added to the chromosome. In the bee attack, the weight and the agent attack threshold are optimization parameters.

*Fitness Function*

For this exercise, the fitness functions are the described through simulation and plugged into Algorithm 4.2.2. Equation 3.30 defines function $\mathcal{F}_1$ and Equation 3.31 defines function $\mathcal{F}_2$.

$$\mathcal{F}_1 = D_t = \tau_{destroyed} * 100 + \tau_{destruction} * 10 \tag{4.4}$$

$$\mathcal{F}_2 = C_t = \nu_{damage} * 10 \tag{4.5}$$

## *4.3 Implementation into Code*

Implementation approach attempts to maintain good software engineering principles (as outlined in [49]), minimize computational complexity, required resources, and approach the programming with the same SO mind-set as the design. In its original state the design of the program was near monolithic, with rampant entangled dependency, and difficult to follow designs and algorithms. Part of the intent with this research effort is to create a system more independent that could be cross-utilized in many other problem domains.

In order to increase the reusability of the code, there is many endeavors to decouple most of the software package. The focus is to also allow the simulation, Genetic Algorithm and UAV controls structures to perform independently. In order to do this, many separate packages were created. This created seven separate cores: the core mathematics and communication section, the GA backbone, the distributed/parrellel package, the UAV control, the behavior sets, the visualizations and the file management section.

Because the code amounts to well over ten thousands of lines, it is not included, but can be found at $\backslash\backslash FS-afit-29\backslash Enstudents\backslash engstudents\backslash\_Lamont\_Students$. Appendices D and E shows the implementation of the Bee Attack Behavior and DE controller. There is also a user manual and package documentation that outlines usage, design, and known issues, found at the same location.

*4.3.1 SO Implemented.* Because SO decomposition focuses on decomposing the problem into small pieces, it fits with an OO based programming languages like Java. In the attached code (Appendices D, E ) notice the compartmentalization of the code allowing for ease of understanding and reuse. Algorithm 4.1.3 matches the recceTargetArea() method. As do Algorithms 4, 5, and 6 with methods analyzeTargetArea(), moveTowardsTarget(), and getVotingNeighborhood(). The entire bee-attack class matches well with the design with a few modifications to deal with the simulation environment as opposed real world communications.

*4.3.2 Environment.* The software itself is java based. The JDK is java development kit 1.6.0_03. The SDK Eclispe 3.2.2 is used for writing and packaging the software. All run on a Intel Xeon(TM) CPU 2.80Ghz dual core with 2.00 GB of RAM.

The testing itself is conducted on the AFIT High Performance Clusters, as outlined in table 4.3.2.

| Name | Network | Processor Type | Processor (GHz) | CPU Mem (GB) | # CPUs Node | #nodes |
|---|---|---|---|---|---|---|
| Tahoe | GigE | Opteron 248 | 2.2 | 4 | 2 | 64 |
| Provo | GigE | Opteron 248 | 2.2 | 4 | 2 | 16 |
| Banff | GigE | Xeon | 3 | 2 | 2 | 8 |

All of the clusters are Gigabit Ethernet with a crossbar switch.

## *4.4 Chapter Summary*

This chapter discusses the low level design and implementation of the two target engagement rules with algorithm definition. The chapter shows the emergent entangled hierarchy of rules and the implementation of a dynamics synergistic controller. The structure of SOGA algorithm is described. This indicated that the emergent entangled hierarchy only depends on the operators and their interaction, independent from the lowest level implementation. Each of the SOGA operators are shown in standard algorithm format. Finally, the application and optimization of the new SO rule set to SOGA is shown. Each of these four aspects of the design, migration, bee inspired attack, DE Controller and SOGA are tested as described in the next Chapter.

# V. Design of Experiments

This Chapter defines the experiments needed to support the validity of each of the major objectives of research. Focusing initially on comparisons to known benchmarks, primarily from [59]. As the system is extended the pieces build successively upon the validation of the previous step. Various scenarios are used to validate the distributed entangled architecture. It starts by laying out a full testing schedule. Then the description of the SOGA and new behaviors and emergent control structures is outlined. Finally the introduction of a new set of attack specific tests are reviewed.

*5.0.1 Computational Experimental Development.* In order to address the complex question of which the MOEA and non-deterministic control sets perform the best we turn to Barr [9]. There are four questions that are raised:

- What is the best solution found? (Effectiveness)
- How long does it take to determine the best solution? (Efficiency)
- How quickly does the algorithm find a good solution? (Efficiency)
- How robust is the method? (Effectiveness)

The answer for the first question, when analyzing an MOEA, comes in the form of a Pareto Front [20]. This is the set of solutions that represent the trade-offs in the objective set. The objectives in our problem domain are minimized casualties and maximized damage, as a result the $\mathcal{PF}$ is two dimensional, see Chapter III. Although effectiveness is important the second question is more relative. Any EA, or non-deterministic optimizer, produces results orders of magnitudes quicker than its linear search counterparts [16]. Time comparison amongst EAs is more a factor of probabilistic nature of the solution space, parameters and tolerances than capacity. Given that the solution space and computational complexity are $O(n^{t_p m^2})$ and the system uses a polynomial non-deterministic algorithm to optimize the best solution may not be found. Finding "good" solutions is what non-deterministic solvers do. Finding good solutions may not vary much among the investigated approaches, empirical data did not show large differences in the run time of the algorithms. Also,

after a small number of generations the algorithm may lose it effectiveness, because of the complexity and probabilistic nature of any single scenario. In order to address a variety of increasingly difficult scenarios are utilized for testing. The result of which should not effect the feasibility of the solutions. The final question, robustness, is an important test in this case as one desires solutions that allow for successful employment in unknown and dynamic situations.

For these reasons we focus the testing of question one related directly to the capabilities of the multi-objective genetic algorithms. The second point of interest, question four focuses on the output of the control set developed. This is evaluated through statistical analysis and visualizations. Each of these aspects is evaluated in every aspect of the simulation.

First the backbone of the GA are validated against the bitGA used by Price [59] and then NSGAII [25] is used to validate successful application to the MOP domain. Testing of SOGA is compared against the performance of the established MOEA, NSGAII. Each of the additional target engagement rules sets are validated through similar comparison and visualization. Finally the optimization of the overall behaviors sets and improved control structure are tested and "optimized". All aspects of the testing are illustrated in Table 5.1. Here questions objectives 1, 2, 3, 4, 5, 6, and 7 all address Barr's first question and objectives 2, 5, 7, and 8 address Barr's fourth question.

## *5.1 Testing Statistics*

For the purposes of being statistically sound the testing ratio for the simulations follows the Central Limit Theorem [44]. For this reason any testing the need to be validated through the CLT has a population of 30 or more. In statistically evaluating the results two issues must be addressed. First their is no assurance that the data is a normal distribution, so to show independence of the populations a Kruskal-Wallis test is used. Secondly the best solutions are not known, so there is no $\mathcal{PF}true$. That

Table 5.1: Testing Outline

| Objective Number | Section | Testing Initiative |
|---|---|---|
| 1 | SOGA | Validation against bitGA |
| 2 | ” | Statistical equivalence with NSGAII |
| 3 | Migration | Statistical dominance of previous behavior set |
| 4 | Bee Attack | Validity in problem domain |
| 5 | ” | Statistical dominance of previous BS with Migration |
| 6 | DE Control Structure | Validity in problem domain |
| 7 | ” | Statistical dominance of Neural Networks |
| 8 | ” | Enhanced operation in new Attack Optimization Framework |

means all indicators of dominance not utilize a known front. Hypervolume and epsilon indicators were chose from the list of possible indicators in [19] for this reason.

## *5.2 SOGA*

The experimentation focuses on comparison of the SOGA implementation, from Section 3.3, to the original GA implementation found in [59]. The objective is to validate that the SOGA algorithm works as well as known robust MOEAs, for example NSGAII, without parameter tuning. Objectives 1 and 2 are from table 5.1. For this reason the system reutilizes a lot of the information gained from the previous work [59]. Heterogenous swarms were shown to be more effective. For the validation of the new MOGA, the experiments utilizes previous heterogeneous swarm testing scenarios setup as a benchmark [59]. Also defined where the optimal swarm sizes in this domain to be between 10 and 30 UAVs .

Given this basic parameter set, SOGA accomplishes the same set of tests. NSGAII also attempts to optimize in this domain. These two approaches are then compared to the bitGA results.

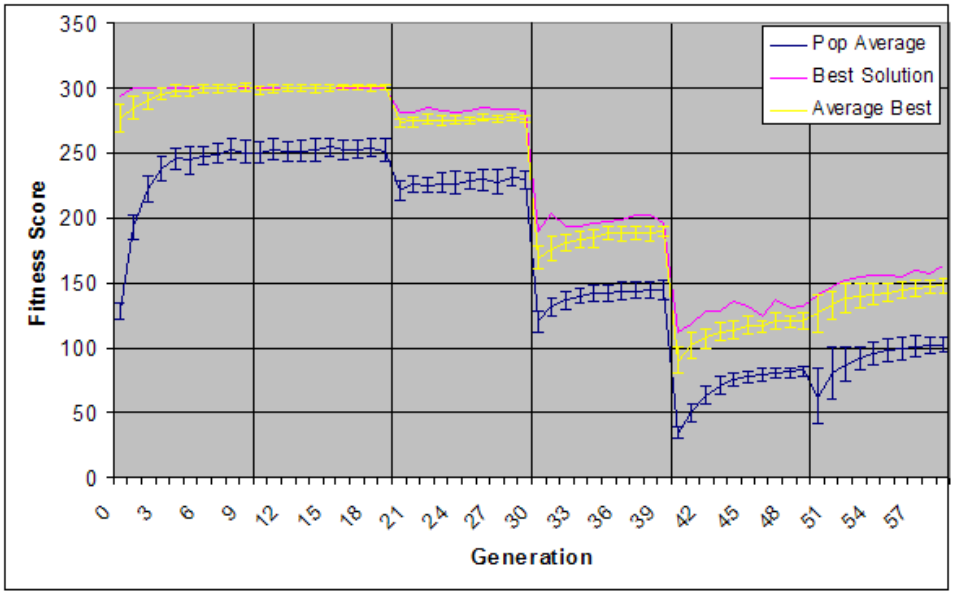


Figure 5.1: Single Objective Mean and Best score over 60 generations with a population of 100 for five iterations with 50 simulations for each fitness calculation. Vertical lines mark changes in the scheduled scenario. Mean score standard deviation is indicated by the intervals [59].

*5.2.1 Metrics.* This aspect of the testing provides "optimization" curves for the success of the swarm in a given environment. The comparison of the GA uses the optimization graph shown in Figure 5.1. Note the original implementation only optimizes on one variable, however there are two variables for comparison. The damage rate inflicted by the swarm and the casualty levels are not directly dependent variables. Translating the optimization rates takes insight to the problem domain and can be further understood through the visualization. In the visualization the swarm needs to exhibit behaviors of more defined attacks, reduced collisions, and optimized intra-swarm movement.

Price's work [59] did not include the effects of lost agents. The inclusion of the factor of safety to the agents draws that fitness function lower in range. These results in Figure 6.6 show similar trends as Price's work [59] shown in Figure 5.1. The benefit of the Multi-Objective Optimization Problem (MOP) is seen in the later generations. As the scenarios get harder the gene takes less time to reach convergence. This can be attributed to the higher strength of the genes (building blocks) at the beginning of

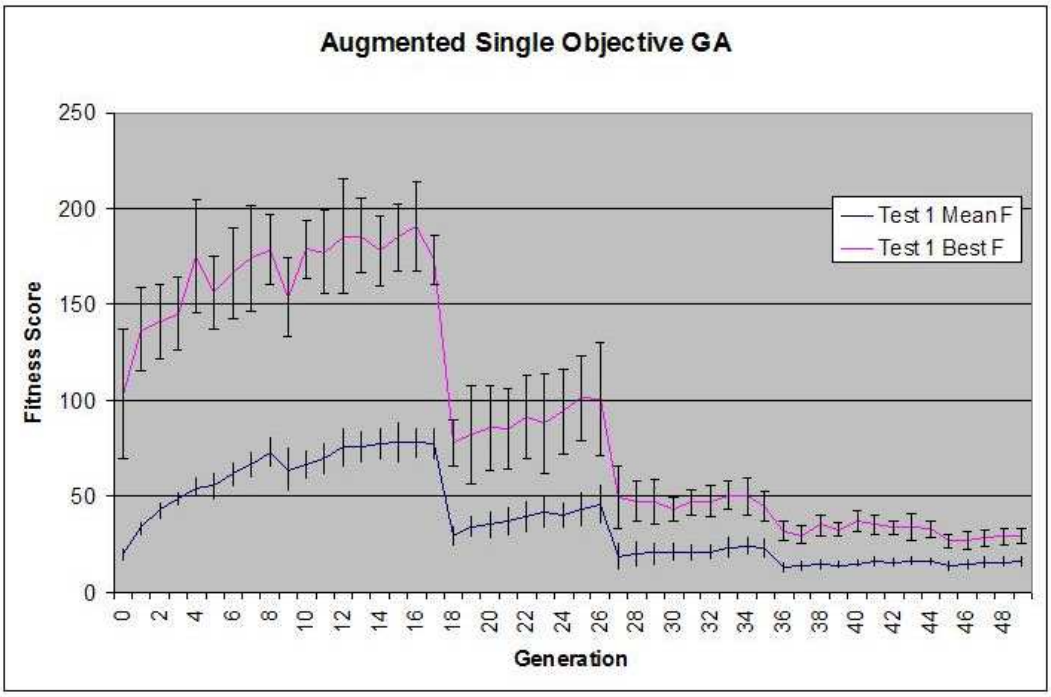


Figure 5.2: In this graph bitGA ran 5 iterations of 30 simulations per fitness function over 50 generations with a population of 100 individuals. Six scenario files are used that increased the difficulty of the evaluation.

those scenarios. Because of the added complexity with this new fitness function more thorough testing is needed to establish fully optimized solutions. This augmented fitness function provides better insight to MOP formation.

Between the two MOEAs several more comparisons are possible. The populations are compared using the Kruskal Wallis P-Values (KW-P), $\epsilon$-indicator, and HyperVolume. The KW P-values given the significance of a population without making the assumption that the distributions are normal. The two indicators measure MOEA effectiveness without having a the best known Pareto Front. This series of

statistical analysis shows the difference and indicate the strength of the better solutions.

*5.2.2 Simulation Environment.* In all of the simulations used during optimization testing, the problem domain remains relatively similar. The basic dimensions are 800km x 800km, keeping the simulations the same size for benchmarking. There are no obstacles in the domain. The ratio of UAVs to targets are at 10:3. If an agents leaves the simulation environment it can not return and is considered dead. Targets do not move. Figure 5.3 shows, generically, the simulation landscape. In Figure 5.3 blue are representative of agents and red of targets. The circle describe the notional detection and engagement ranges around bother targets and UAVs. The weights show the strength of the associate agent and the radial line on the UAV agents show the movement direction.

*Agent Setup* Agents differ slightly in heterogeneous swarms. The kinematics of the agents have all the same since fidelity in this stimulation. This does not warrant more exacting physics models because the implementations here are not geared to any specific hardware. The agents are all loosely based on the model of a reaper UAV [59]. Table 5.2 shows the original specifications of the agents. Note there is a difference between the UAV (sensor agents) and UCAVs (attack agents) which theoretically use the same airframe but differ in payload packages.

Table 5.2: Original Agent Parameter Settings

| Parameter | Value |
|---|---|
| Max Velocity | .07716 $\frac{km}{sec}$ |
| Hit Points | 10 |
| Max Damage | 1 $\frac{point}{sec}$ |
| Communication Range | 1.5Km UCAV (10km UAV) |
| Detection Range | 1.5Km UCAV (10km UAV) |
| Engagement Range | 1 Km UCAV (none for UAV) |
| Behavior Archetypes | 3 |

Table 5.3 shows the behavior set used during all three testing stages in this section.

Table 5.3: Original Agent Behavior Set

| Behavior Rule |
|---|
| Flat Align |
| Cohesion |
| Repulsion |
| Weighted Target Orbit |
| Weighted Target Attract |
| Flat Target Align |
| Weighted Target Repel |
| Flat Target Repel |
| Evade |
| Avoid Obstacle |

*Target Setup* The target parameters stay static throughout the simulations. The targets draw from the same object set in implementation as the agents but do not move and have different detection and engagement rings. Table 5.4 shows the set up used for the targets. This simulates ground based SAM sites.

Table 5.4: Original Target Parameter Settings

| Parameter | Value |
|---|---|
| Max Velocity | 0 |
| Hit Points | 10 |
| Max Damage | 1 $\frac{point}{sec}$ |
| Communication Range | 10Km |
| Detection Range | 10Km |
| Engagement Range | 2 Km |

Table 5.5: Genetic Algorithm Testing Structure

| Parameter | Value |
|---|---|
| Sims per Eval | 30 |
| Generations | 60 |
| Generations per Epoch | 10 |
| Runs of full set of Gens | 5 |
| Population Size | 60-100 |

*5.2.3 GA Setup.* The chromosome described in section 2.5.2 maps the values to the control arbiter for the BAs. The remaining chromosome represents the control

weightings of the rules themselves. Each chromosome gets evaluated 30/times on the same scenario, with slight variations in location and starting headings of agents and targets (location only). Each generation carries between 60-100 chromosomes. The system runs through 60/generations with a change in the simulation scenario at the epoch of every 10th generation. Empirical analysis indicates that the system starts to converge within 5-6 generations. The increased difficult after 10 generation allows the system to optimize in more difficult scenarios without strong casting the controls before moving on to a more difficult scenario. These sixty generations are run 5 times to establish an average for each generation and comparison to the other MOGA approaches. The system uses 5 iterations to balance between computational time and statistical accuracy. With the addition of a variance calculation it is reasonable to assume an understanding of the systems behavior has been reached when the variance is small. Table 5.5 outlines the structure of the GA parameters.

*BitGA* The parameters for the bitGA are from Price's [59] benchmark work. These are the same parameters used during the augmented fitness function test. Table 5.6 shows the parameters used.

Table 5.6: bitGA Testing Parameters

| Parameter | Value |
|---|---|
| Population | 100 |
| Preserve Pop | 60 |
| Crossover Rate | .1 |
| Mutation Rate | $\frac{1}{C_r}$ |
| Mutation Neighborhood | .05 |

*NSGAII* Table 5.7 presents the parameter values used. Through empirical testing the best parameter values are narrowed to a small window. The variance of results in this window do not show significance therefore average values are used to run NSGA-II optimization.

*SOGA* Table 5.8 shows the parameter set used for the SOGA testing. The population sizes are trivial, but the use of a small random population every generation does improve performance by continually injecting exploration data. The crossover

Table 5.7: NSGA-II Testing Parameters

| Parameter | Value |
|---|---|
| Population | 80 |
| Preserve Pop | 40 |
| Crossover Rate | .1 |
| Mutation Rate | $\frac{1}{C_r}$ |
| Mutation Neighborhood | .05 |

rate is self adaptive and defines the mutation rate. The mutation neighborhood is self adaptive as well.

Table 5.8: SOGA Testing Parameters

| Parameter | Value |
|---|---|
| Population | 65 |
| Preserve Pop | 30 |
| Random Pop | 5 |
| Crossover Rate | SA |
| Mutation Rate | $\frac{1}{C_r}$ |
| Mutation Neighborhood | SA |
| CAA rate | $Mn_r$ |

With this testing structure and parameter set the system facilitates analysis between the Genetic Algorithms and validation of the SOGA.

### *5.3 Migration*

This phase tests the addition outlined in Section 3.2.1, migration to chosen waypoints. From objectives 3 in table 5.1, testing has two thrusts: validate that the rules set is working properly and its addition to the overall system performance. The first thrust is accomplished through visualization after the system has been "optimized" in the second thrust.

Testing involved "optimizing" with the same behavior set as the previous test set, Table 5.3, with added migration. Table 5.9 shows the new behavior set. It uses the same GA parameters as in Table 5.8. The analysis of the results focus on the scope of the created chromosomes and the statistical significance of the original population

and the population with the added behavior, outlined in both Section 5.2.1 and testing metrics.

Table 5.9: Agent Behavior Set w/ Migration

| Behavior Rule |
|---|
| Flat Align |
| Cohesion |
| Repulsion |
| Weighted Target Orbit |
| Weighted Target Attract |
| Flat Target Attract |
| Weighted Target Repel |
| Flat Target Repel |
| Evade |
| Avoid Obstacle |
| Migration |

## *5.4 Bee Inspired Attack*

Testing the new bio-inspired attack behavior sets requires a similar approach. Objectives 4 and 5 from table 5.1 are used. Due to the entangled hierarchy, outlined in Section 3.2.1, however, each of the sub-behavior rules must be observed individually. Table 5.10 shows the set of behaviors for this section of testing. Again the GA uses the same parameters as in Table 5.8. Note the sub-behaviors of Bee Attack, indicating the entangled hierarchy present in a bee attack behavior rule set. With the inclusion of the Bee Attack the system favored the passive mode, because reaching ideal conditions for this type of attack is difficult. This reiterates the need for a more adaptive controller.

*Metrics* The same metrics, MOEA indicators and P-Tests, areused to determine the dominance of the different configurations Section 5.2.1, also quantifies the improvement from the added rules sets and resulting emergent behavior functionality.

Table 5.10: Agent Behavior Set w/ Bee Attack

| Behavior Rule |
|---|
| Flat Align |
| Cohesion |
| Repulsion |
| Weighted Target Attract |
| Weighted Target Repel |
| Flat Target Repel |
| Evade |
| Migration |
| Bee Attack |
| Target Recon (*Target Orbit*) |
| Target Set Analysis and Engagement |
| Target engagement(*Flat Attract*) |
| Avoid Obstacle |

## *5.5 Control and Attack Optimization*

To facilitate benchmarking for the control mechanism, basic testing utilizes the same test beds as the previous testing steps, Section 5.2.1. Objectives 6, 7 and 8 from Table 5.1 are addressed. The new control structure is intended to increase the systems ability to work in more dynamic spaces. The normal testing structure only begins to exercise that ability. For this reason the next phase of testing focuses on the probabilistic and dynamic nature of more complex target sets.

*5.5.1 Simulation Environment.* The simulation environment is similar to the previous tests. A few things have been change in the agent/target setup and the scenarios.

*Targets* The target themselves are still stationary but a lot more variability is added to each targets capabilities in order to create more difficult and complex attack situations. Table 5.11 shows the new target parameter ranges.

*Agents* In order to keep the agents on a similar "playing field" the parameters are changed slightly. Table 5.11 shows the new agent parameter ranges. This affords the agents the opportunity to detect the targets before being attacked by them, in

Table 5.11: New Target Parameter Settings

| Parameter | Value |
|---|---|
| Max Velocity | 0 |
| Hit Points | 10 |
| Max Damage | 1 $\frac{point}{sec}$ |
| Communication Range | 3.0 - 10.0Km |
| Detection Range | 3.0-10.0Km |
| Engagement Range | 1.5-3.5 Km |

most cases. This is crucial with the engagement strength of the targets outweighing the agents!

Table 5.12: Original Agent Parameter Settings

| Parameter | Value |
|---|---|
| Max Velocity | .07716 $\frac{km}{sec}$ |
| Hit Points | 10 |
| Max Damage | 1 $\frac{point}{sec}$ |
| Communication Range | 2.5Km UCAV (10km UAV) |
| Detection Range | 2.5Km UCAV (10km UAV) |
| Engagement Range | 1.5 Km UCAV (none for UAV) |
| Behavior Archetypes | 3 |

The agents also choose from new sensor inputs. The previous sensor set included only the swarm density information and the closest target. Here the closest target, waypoint, swarm density, and number of votes for a target are used. This gives the agents the ability to react to many different phases of the attacking sequence.

*5.5.2 Scenario Sets.* The major changes in scenarios relate to the scenario types. The agents are presented with sets of targets that are more diverse and in most cases more "powerful"! The different aspects for which the system tests the agent control structures includes:

- Stronger engagement rings (Overpowering)
- Overlapping targets (Aggregated)
- Groups with increasing individual strength (Building)

- Geographically separated groups (Separate)
- Target Fronts (IADS)

The intent with the first scenario type, Overpowering, is to provide the agent with targets in which the engagement rings outweigh their own detection ranges. Figure 5.5 shows an example of such a testing scenario.

Building on that the next scenario, Aggregated, set utilizes several overlapping targets and sets of different target numbers and strengths. Figure 5.6 shows a scenario with three sets of targets each with its own orientation and strengths.

In the next scenario, Building, the targets in the groups grow in strength, like those shown in Figure 5.7.

In this next scenario, Separate, the targets are grouped but the detection rings do not overlap the others location. Once the system has learned to work with multiple sets, changing the configuration warrants a change in the known abstracted state space. Figure 5.8 shows these separated target groups.

The final new testing scenario, IADS, pits the agents against lines of targets. The intent is to simulate an Integrated Air Defence System(IADS) array. In many cases Aerial Vehicles engage ground targets such as SAM or Radar sites. This tends to be the most difficult and dangerous mission. Figure 5.9 shows a basic IADS setup that reflects one that would be found at the edge of a sovereign air space.

Finally the agents are run against the 6th and most difficult, generic scenario from the original batch again. This allows for application of the trained system in a nonspecific environment. This test also facilitates comparison for the knowledge base. With this set the agents controls are put through their paces without specifically iterating every scenario.

## *5.6 Chapter Summary*

The intent with testing the architecture is to validate all four of the major sub-sets of the research: the GA (SOGA), the behaviors (Migration and Bee In-

spired Attack), and the controller (DE-inspired controller). Each of these sets of tests utilized several methods for analytical comparison: visualization, MOEA indicators (Hypervolume and epsilon), and a Kruskal-Wallis independence test. The testing of objectives 1, 2, 3, 4, 5, 6, and 7 all address Barr's first question and objectives 2, 5, 7, and 8 address Barr's fourth question, thoroughly. The methodical approach to testing used in this section is aimed at creating stable, trustable results in a system UAV swarm that works in dynamic environments with stochastic results and emergent behaviors.

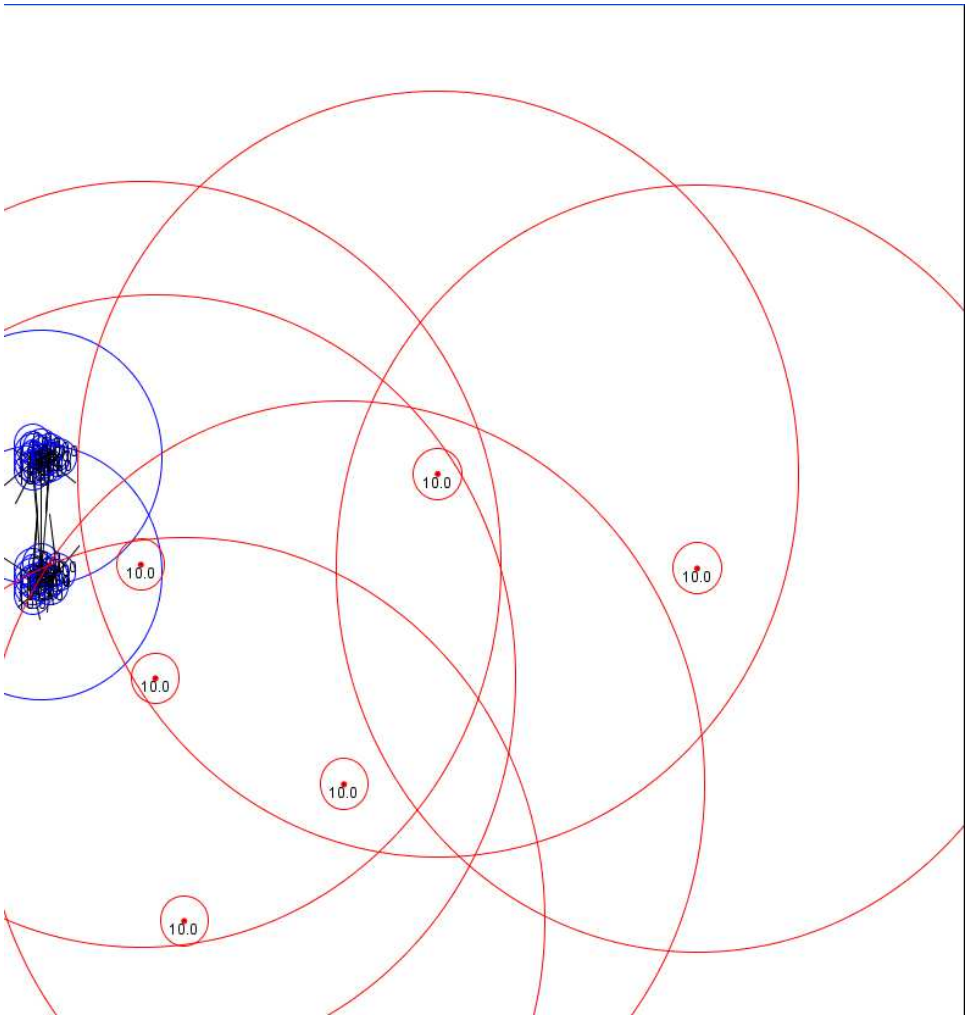


Figure 5.3: This figure gives the basic idea of the initial setup of a simulation. The cluster of points on the left, in blue, are the UAVS, with associated, movement vectors, engagement and detection rings shown. In red the targets are shown with target strength, detection and engagement rings shown around as well.

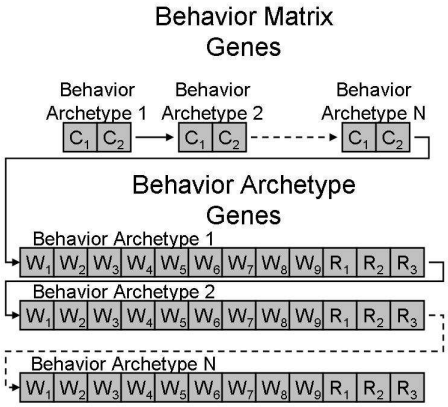


Figure 5.4: There is a connection weight for each sense for each behavior archetype. These are followed by 12 genes which describe the weights and radii for the behavior rules for each behavior archetype.

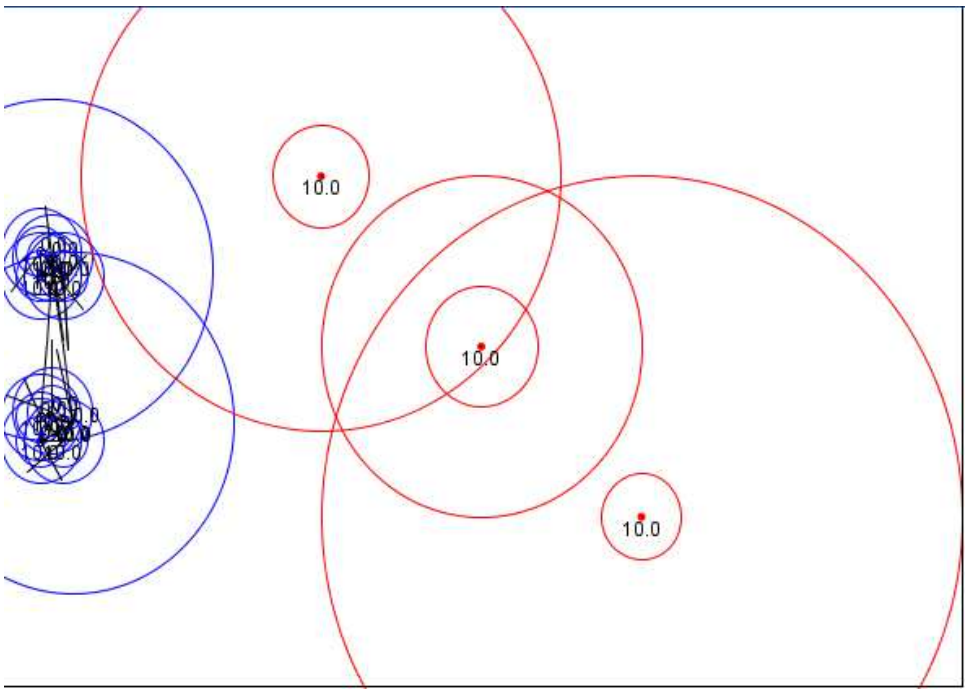


Figure 5.5: Example strong stand-alone targets.

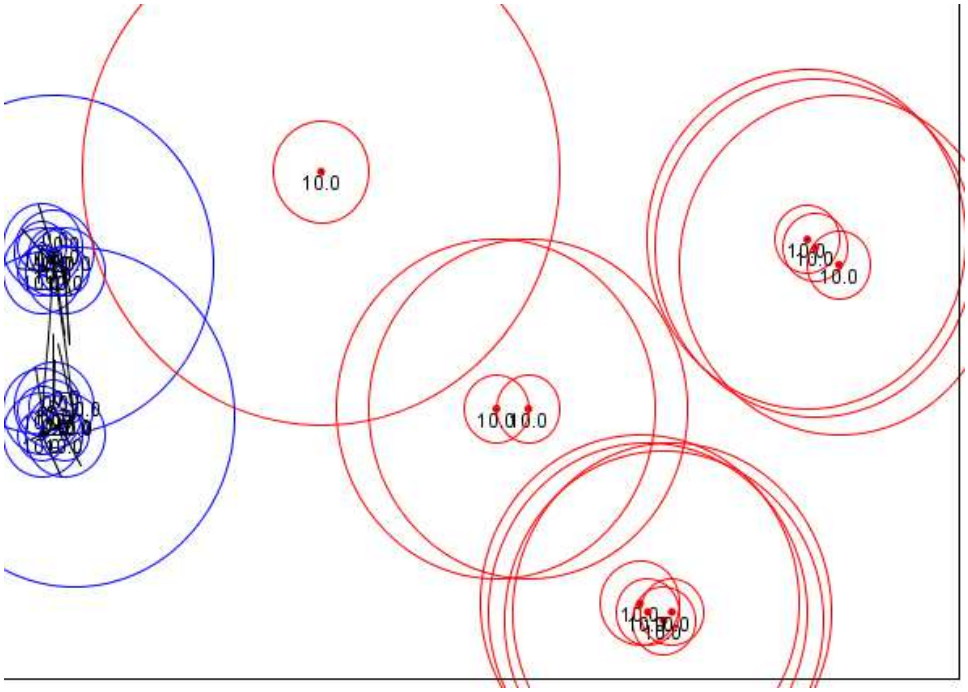


Figure 5.6: Example scenario with several sets of differing overlapping aggregated target strengths.

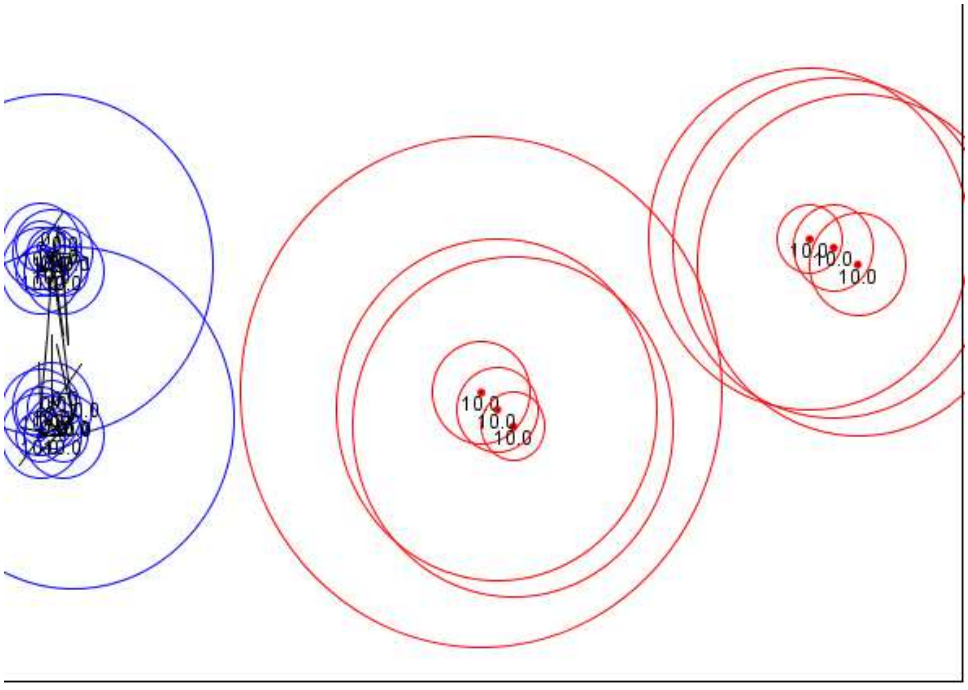


Figure 5.7: Example of groups of growing strength.

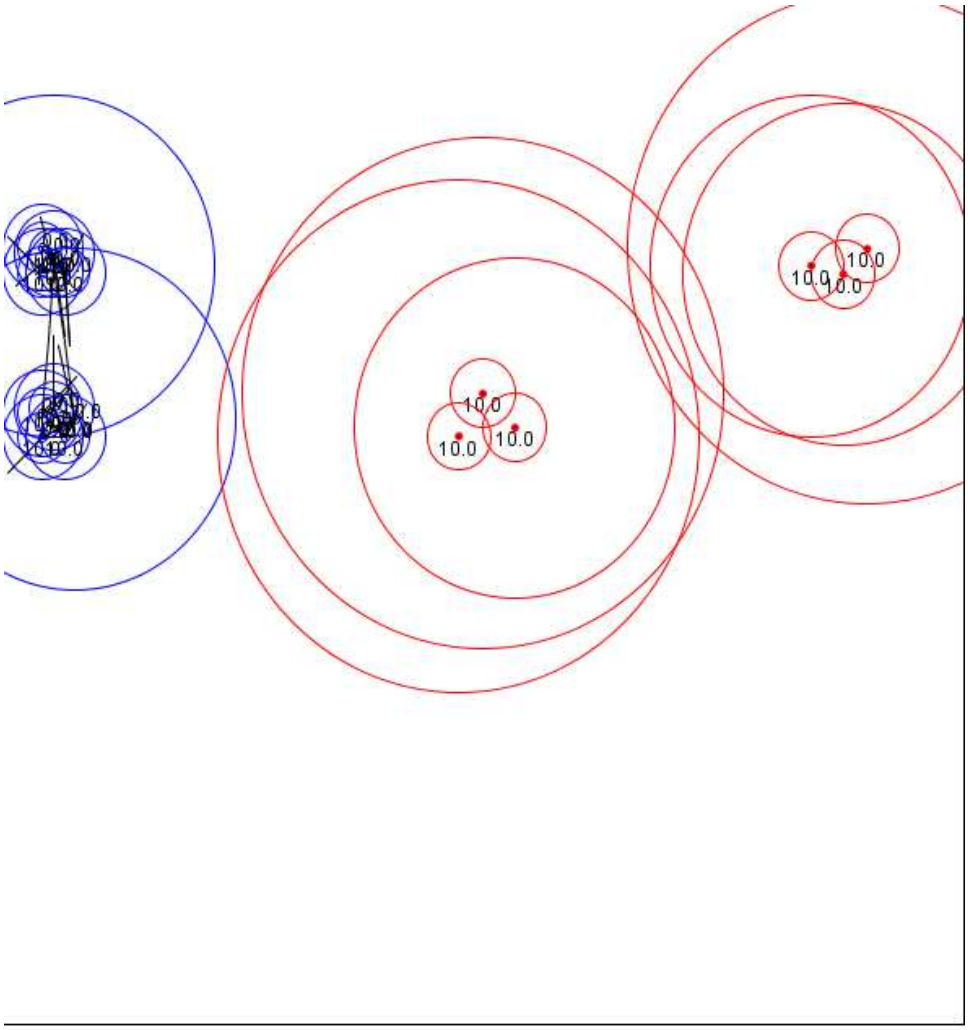


Figure 5.8: Example of separated groups.

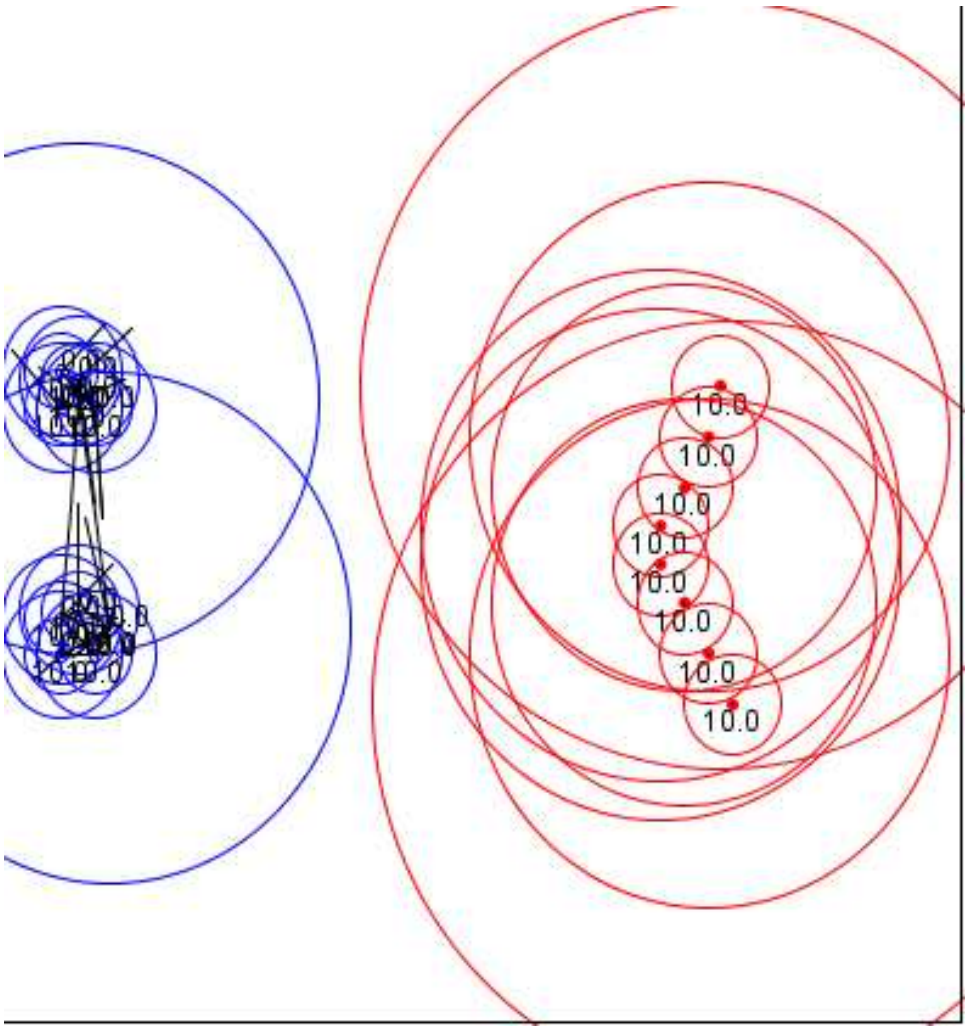


Figure 5.9: Example of IADS front scenario.

# VI. Analysis of Results

Analysis of the testing data flows from the design of experiments in Section V. Various Tables and graphs are presented with statistical analysis that indicate a very successful development of UAV swarm combat behavior using the prescribed architecture. Table 6.1 shows the testing schedule and results summary. Sections in this chapter that discuss the results are also indicated. Overall the experiments completed the assigned tasks, which are "optimized" for successful engagement in target environments. In general, the computational system creates a more effective behavior and out performed the original configuration [59]. As outlined in Table 6.1, this chapter starts with a short discussion of the SOGA development. Once the validity of SOGA is established, results from each of the additions to the controls and behaviors of UAVs is articulated. Finally the performance of the resulting system and the original configuration is evaluated.

Table 6.1: Results Schedule

| Objective Number | Section | Testing Initiative | Results |
|---|---|---|---|
| 1 | SOGA | Validation against bitGA | Derived - Section 6.1.3 |
| 2 | " | Statistical equivalence wtih NSGAII | Achieved - Section 6.1.3 |
| 3 | Migration | Statistical dominance over previous behavior set | Achieved - Section 6.2 |
| 4 | Bee Attack | Validity in problem domain | Visualization - Section 6.3.1 |
| 5 | " | Statistical dominance of previous BS with Migration | Achieved - Section 6.3.2 |
| 6 | DE Control Structure | Validity in problem domain | Achieved - Section 6.4.1 |
| 7 | " | Statistical dominance over Neural Networks | Not Achieved - Section 6.4.1 |
| 8 | " | Enhanced operation in new Attack Optimization Framework | Achieved 6.4.2 |

## 6.1 MOEA

The results from the initial set of tests indicate the effectiveness of the SOGA algorithm. The NSGA-II data servers as the benchmark MOEA for the development of SOGA feasibility. A Monte Carlo approach illustrates the scope of the domain. The data from the augmented BitGA shows the relationship between the single objective and multi-objective domains.

*6.1.1 NSGA-II.* Figure 6.1 shows the carry over population $\lambda$ for the "full" run of the system over the course of 5 iterations of 60 generations. The changes in shape show the changes between each scenario file. For each of the scenarios the population pushes to the edges of the $\mathcal{PF}_{known}$.

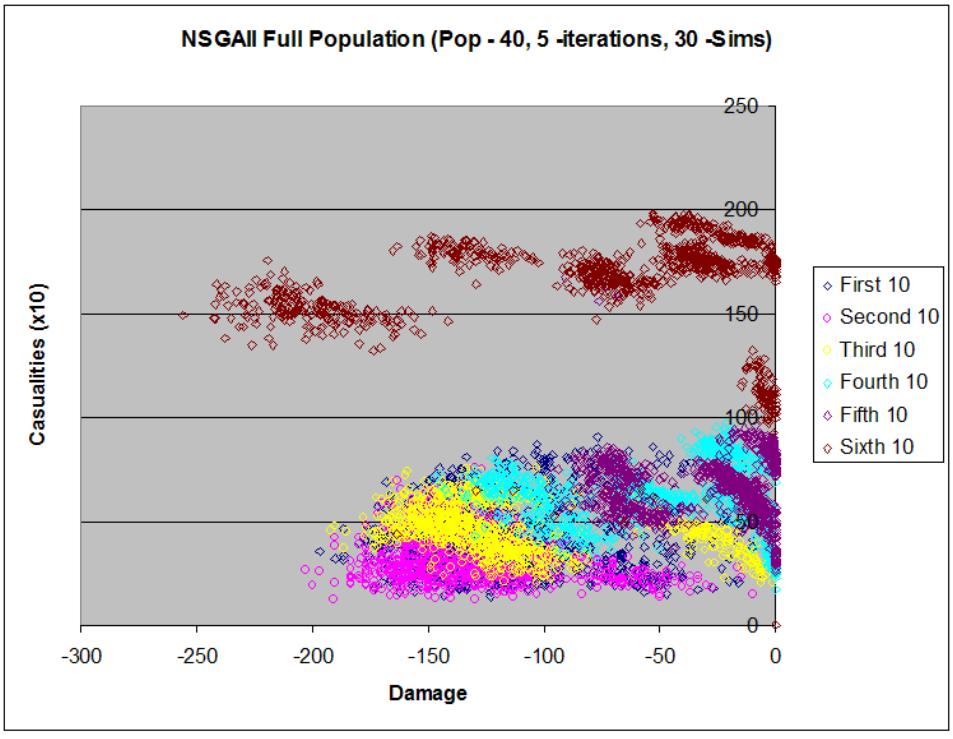


Figure 6.1: In this graph NSGA-II ran 5 iterations of 30 simulations per fitness function over 60 generations with a population of 40 individuals. Six scenario files are used that increased the difficulty of the evaluation.

The learning rates, shown in Figure 6.2, indicated the movement of the solutions towards the fronts of the solution space. The standard deviations grow as the scenario files get more difficult.

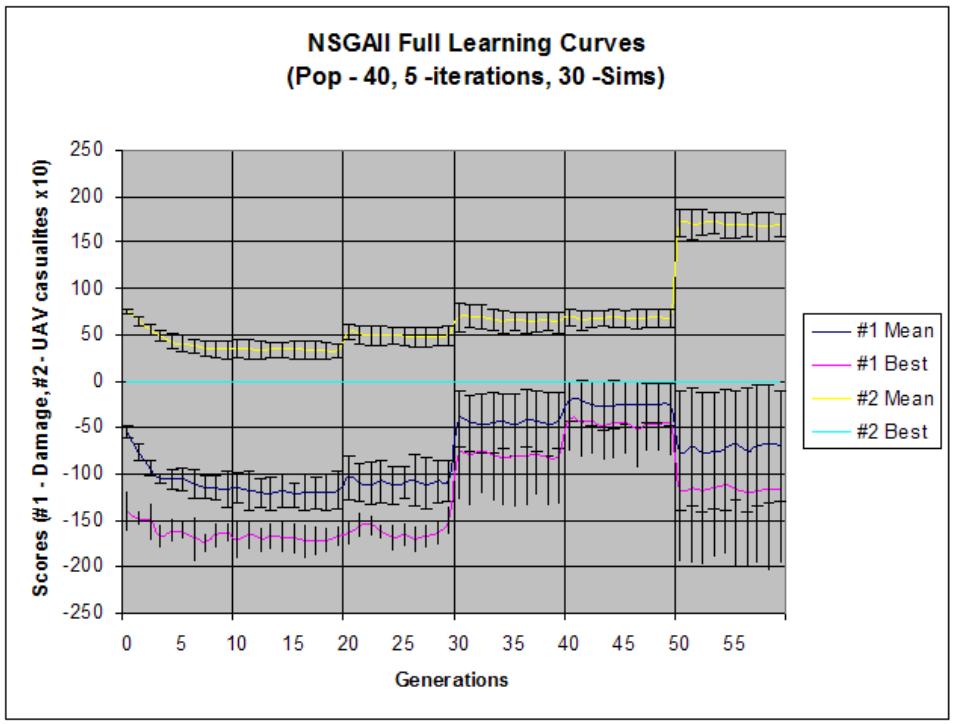


Figure 6.2: Mean and Best score by 40 generation with 60 populations for five iterations with 30 simulations for all runs. Vertical lines mark changes in the scheduled scenario. Mean score standard deviation is indicated by the intervals

*6.1.2 SOGA.* Figure 6.3 represents the SOGA population the same way as in Figure 6.1. The population shows fairly even distribution over the $\mathcal{PF}_{known}$.

The learning rates, shown in Figure 6.4, for SOGA are similar the previous works. The standard deviations are not as large but still increase in the more difficult scenarios.

*6.1.3 Comparison of SOGA Against Benchmarks. Monte Carlo* Figure 6.5 shows a Monte Carlo simulation for the easiest scenario. Note the population includes only the top 20 of 40 random selected Monte Carlo simulation points. The objective of this figure is to compare, visually, that the NSGA-II and SOGA algorithms achieve

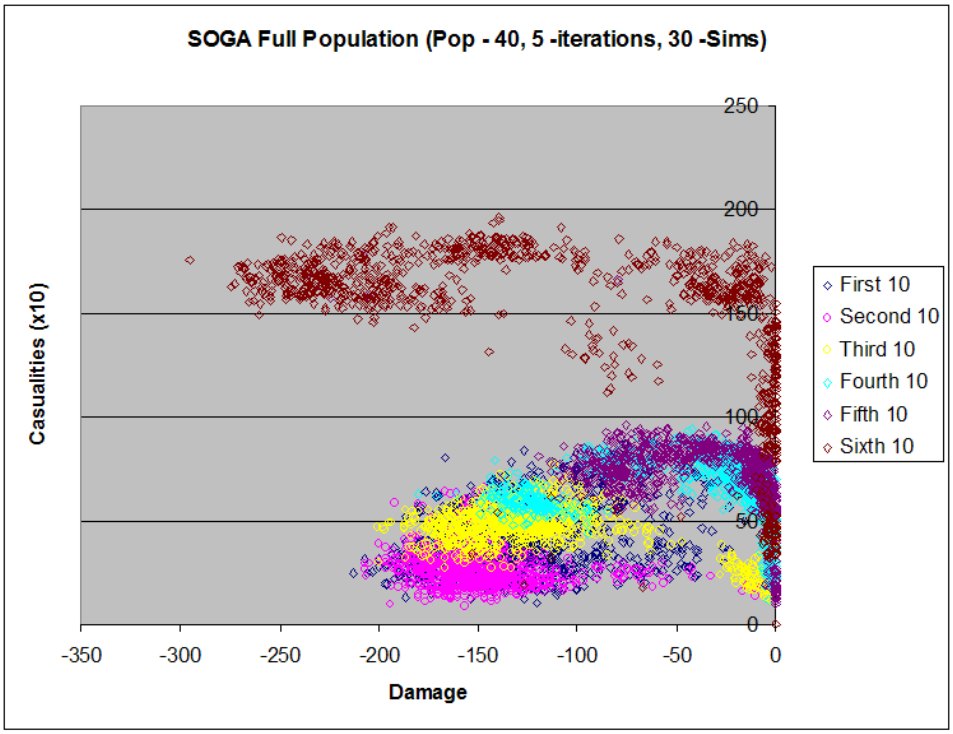


Figure 6.3: In this graph SOGA ran 5 iterations of 30 simulations per fitness function over 60 generations with a population of 40 individuals. Six scenario files are used that increased the difficulty of the evaluation.

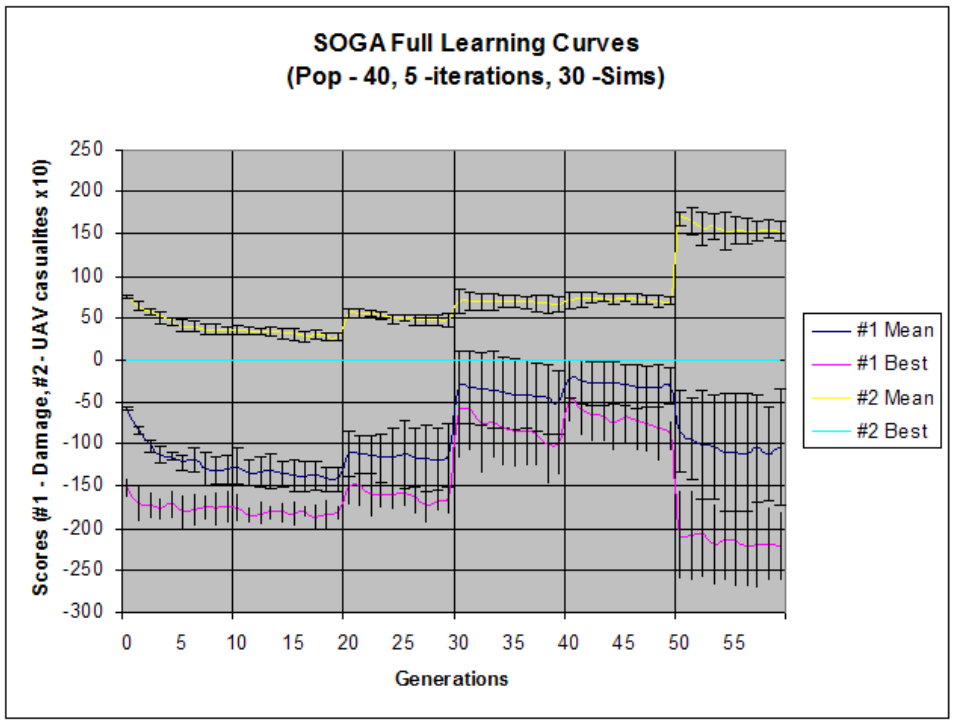


Figure 6.4: Mean and Best score by 60 generation with 40 populations for five iterations with 30 simulations for all runs. Vertical lines mark changes in the scheduled scenario. Mean score standard deviation is indicated by the intervals

better known Pareto front solutions than a random simulation. When compared with Figures 6.1 and 6.3 it is seen that the Monte Carlo does create solutions that are near the $\mathcal{PF}_{known}$. Upon inspection of the graphs, the center of mass of the Monte Carlo population, however, falls behind the other two algorithms. This shows the pressure and manipulation of the NSGA-II and SOGA algorithms force the front toward better solutions.

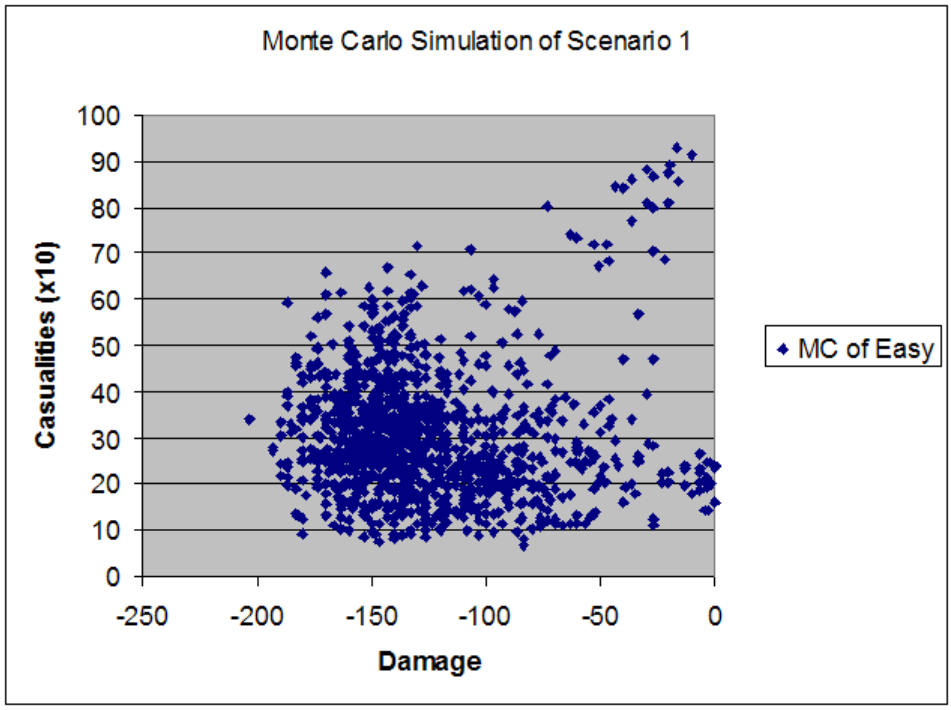


Figure 6.5: Monte Carlo simulation with selection of the best population 30 generations with 60 population.

*Bit GA*

The MOEA solutions are compared to the previous objective space shown in Figure 6.6. There exist no direct correlation between single dimensional solutions and vectors in multi-objective space. Figures 6.2 and 6.4, however, clearly illustrate higher destruction levels in the final scenarios, using the MOP approach. This is only one indicator of the increased strengths, more are shown in the next section.

*Single Scenario Learning*

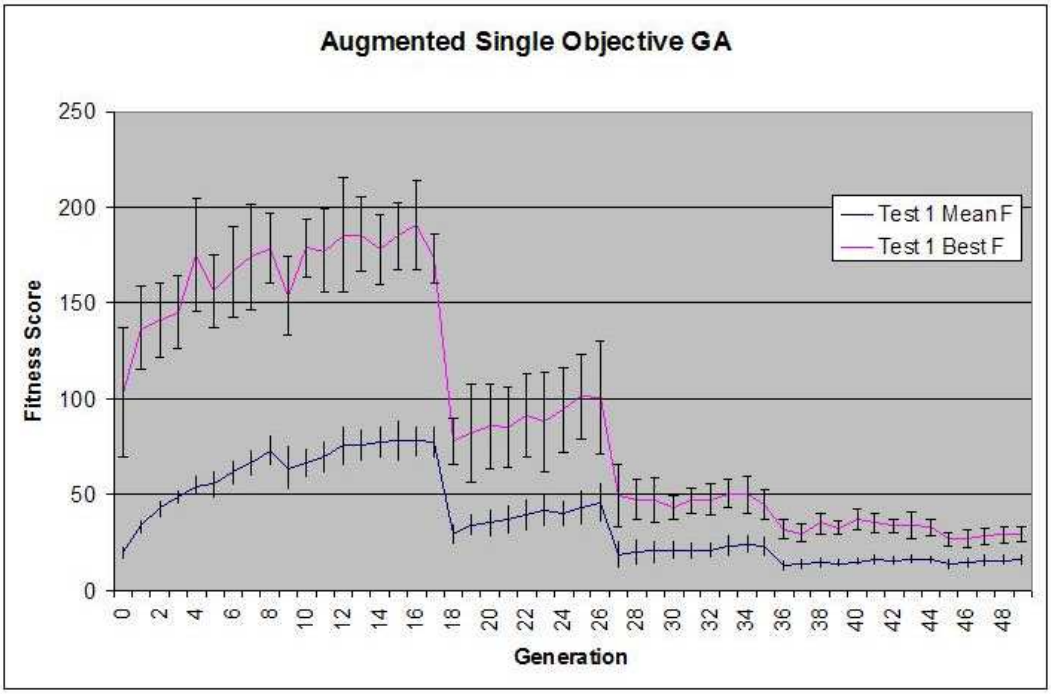


Figure 6.6: In this graph bitGA ran 5 iterations of 5 simulations per fitness function over 50 generations with a population of 100 individuals. Six scenario files are used that increased the difficulty of the evaluation.

Figures 6.7 and 6.8, are another indication of the strength of a MOEA implementation. This shows the ability of the MOEA to find solutions in the harder scenarios, because they do not require the stepwise approach of increasing strength scenarios. This shortfall in the original implementation of bitGA was noted in [59]. In this test the population grew on a single difficult scenario file, as opposed to growing the population through increasingly more difficult scenarios (as shown in [59]), with the same effectiveness.

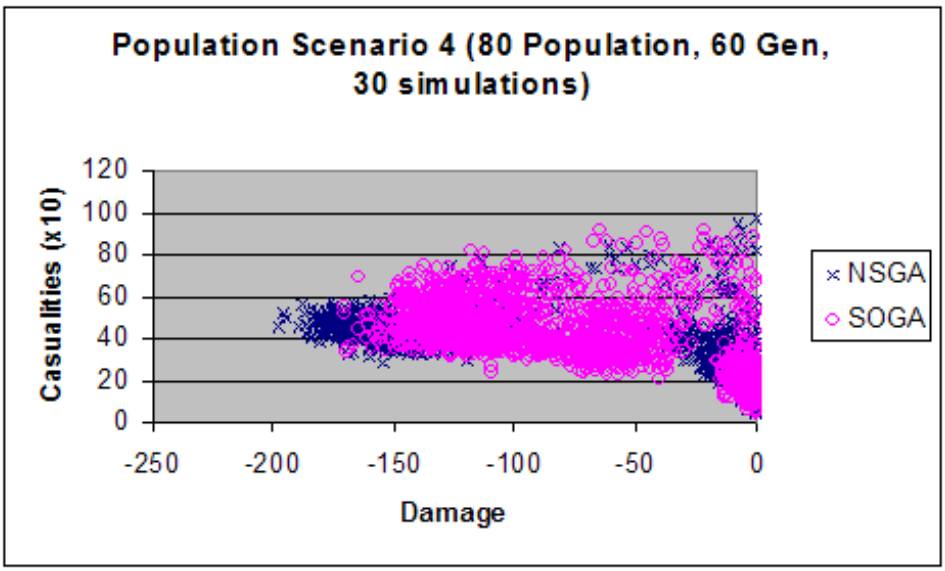


Figure 6.7: This graph shows the population for both GAs run 60 generations with 30 simulations per fitness function over a population of 80 individuals on the 4th scenario.

*SOGA vs. NSGAII*

The testing over all six scenario files paints a clearer picture. Figure 6.9 compares $\mathcal{PF}_{known}$ of the algorithms. Sets of similar points show the difference in the two algorithms on each scenario. Upon inspection there is a noted difference in the results favoring SOGA.

Table 6.2 compares the results of every generation over every iteration for each scenario file on each objective using the Kruskal-Wallis test. A p-value of less than 0.05

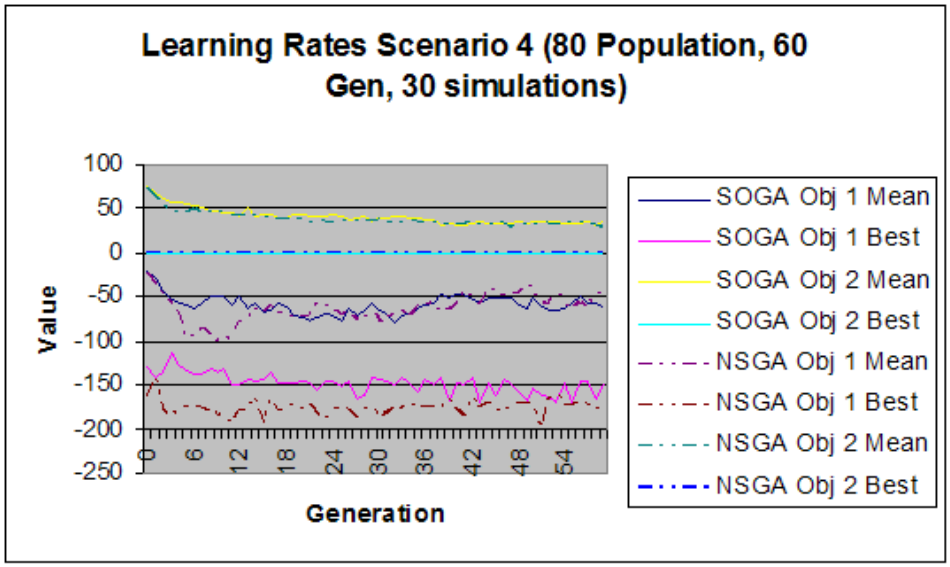


Figure 6.8: This graph shows the learning curves for both GAs run 60 generations with 30 simulations per fitness function over a population of 80 individuals on the 4th scenario.

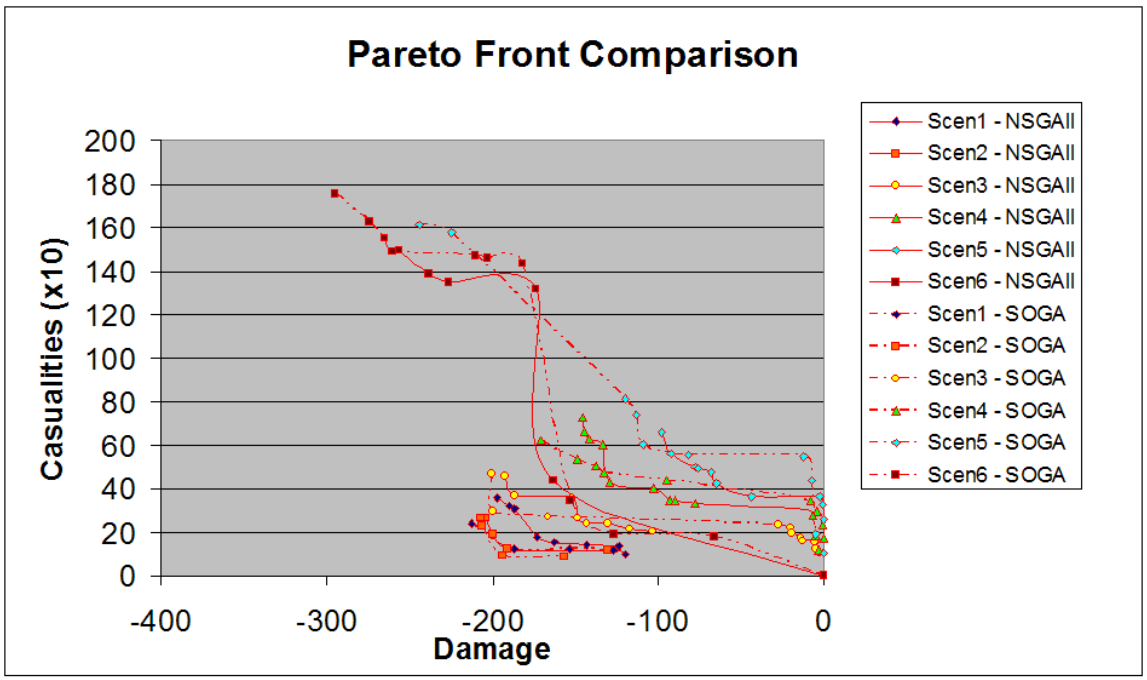


Figure 6.9: This graph compares the $\mathcal{PF}known$ of scenarios. NSGA-II shown with solid lines and SOGA shown with dashed lines. Each scenario level has the same point symbol for both.

indicates a false null hypothesis, so all of the indicated distributions with significance show that the SOGA population has better values.

| Scenario | Objective Damage | Objective Casualties |
|---|---|---|
| 1 | .4647 | .9168 |
| 2 | .1172 | .1172 |
| 3 | .0494* | .9168 |
| 4 | .3472 | .754 |
| 5 | .6015 | .4647 |
| 6 | .4647 | .0758 |

Table 6.2: Kruskal-Wallis P-Value SOGA vs NSGA-II
* - in most cases SOGA outperforms NSGA-II in this case it is significant

Both of the MOEA indicators show in Tables 6.3 and 6.4 show the same trends. SOGA does reach points not generated by NSGA-II, in some instances. In general, however, these two indicators show the slight dominance of SOGA.

| Scenario | SOGA-NSGAII | NSGAII-SOGA |
|---|---|---|
| 1 | 0 | 15.6 |
| 2 | 0 | 3.3 |
| 3 | 6.3 | 7.3 |
| 4 | 10.6 | 10.6 |
| 5 | 18.4 | 95 |
| 6 | 16.3 | 25.3 |

Table 6.3: $\epsilon$-Indicator for SOGA/NSGAII

Hypervolume calculations use the reference point of (0,200). In all scenarios there are twenty UAVs (x 10 scaling) which is why 200 was used. Zero damage is the minimum value of that objective. As a result the Hypervolume calculations agree with what visual inspection of Figure 6.9 indicates, in most case SOGA dominates, but is inconclusive overall.

All of the indicators, statistical tests and analysis show that optimization by SOGA is not statistically different from those produced by NSGA-II. This meets the testing objective number 2. It also means that this GA without parameter tuning

| Scenario | SOGA | NSGAII |
|---|---|---|
| 1 | 39871.552 | 33257.4 |
| 2 | 39334.54 | 38049.22 |
| 3 | 34729.26 | 33712.61 |
| 4 | 26172.31 | 23443.28 |
| 5 | 22354.73 | 15276.4 |
| 6 | 34132.44 | 31301.71 |

Table 6.4: Hypervolume for SOGA/NSGAII

can perform as well as one of the known standards in this domain. The rest of the research builds on this fact.

### *6.2 Migration*

The objective for migration is to improve performance over the previous behavior set, Section 6.1.3. For this reason, the results for migration are gathered in the same way as the SOGA results in the previous section. Comparison of the optimization (or "learning") rates and populations are performed through the graphical and statistical analysis.

Figure 6.10 shows populations for all of the six scenarios. The population forms in a similar manner as the previous tests. To differentiate the two setups, analysis of the population $\mathcal{PF}known$ and population distribution are used.

The optimization curves, in Figure 6.11, look similar to those in the previous test, Figure 6.4. Looking at generations 50 and on allows for comparison of the final state of the system. Although the best performances are approximatively equal to the averages, they out perform the previous ones by a small margin, shown in further analysis.

Graphical analysis of the population is depicted in Figure 6.12. The original setup and that with migration added are compared through $\mathcal{PF}known$ for each scenario. In almost every case the $\mathcal{PF}known$ for the migration setup moves further to the lower left of the graph. Statistical analysis of these fronts is again done with

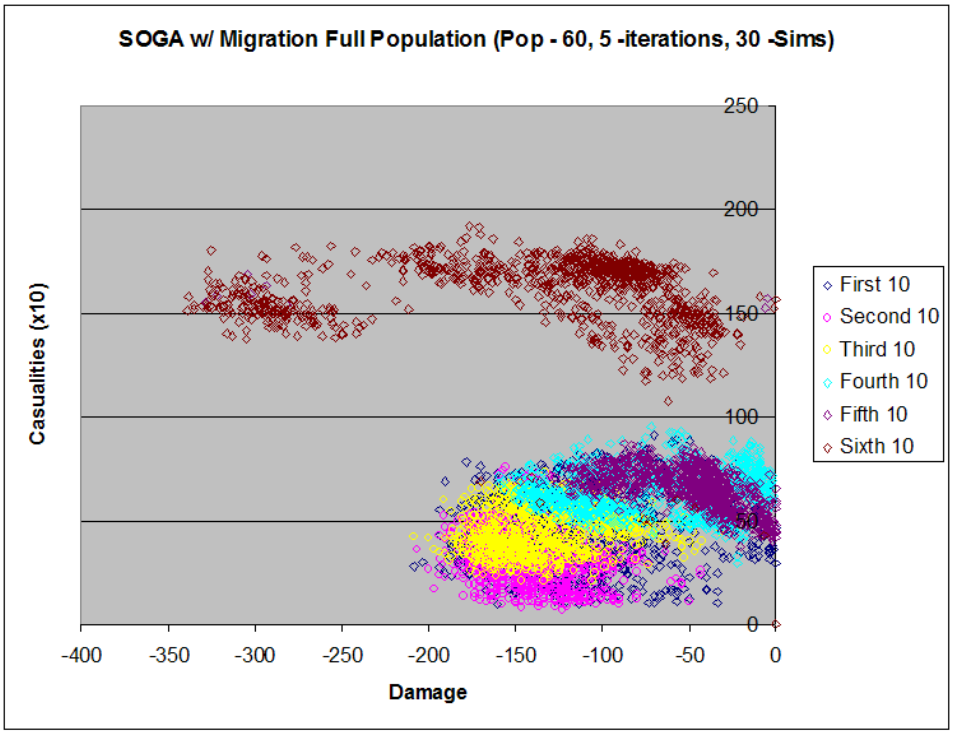


Figure 6.10: In this graph SOGA, using the base rules and Migration, ran 5 iterations of 30 simulations per fitness function over 60 generations with a population of 60 individuals. Six scenario files are used that increased the difficulty of the evaluation.

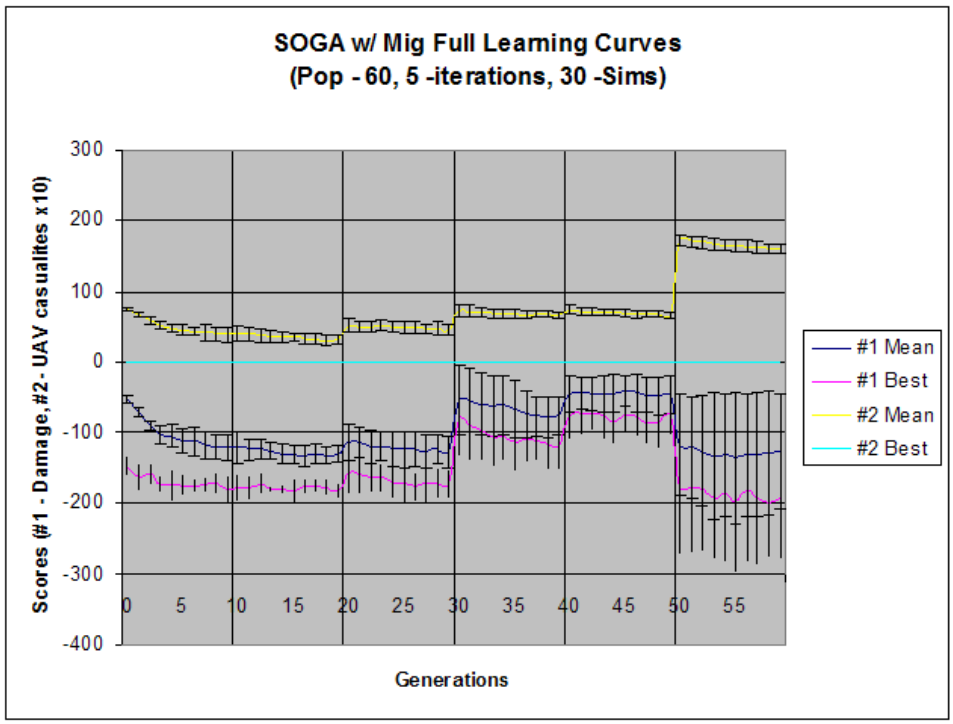


Figure 6.11: The base rules set are used with the Migration rules added. Mean and Best score by 60 generation with 60 populations for five iterations with 30 simulations for all runs. Vertical lines mark changes in the scheduled scenario. Mean score standard deviation is indicated by the intervals.

Hypervolume and $\epsilon$-Indicator metrics. Table 6.5 shows the hypervolume calculations for this comparison. In 4 of the six cases the Hypervolume of the migration exceeds the first, including the last scenario. This is an indication of dominance.

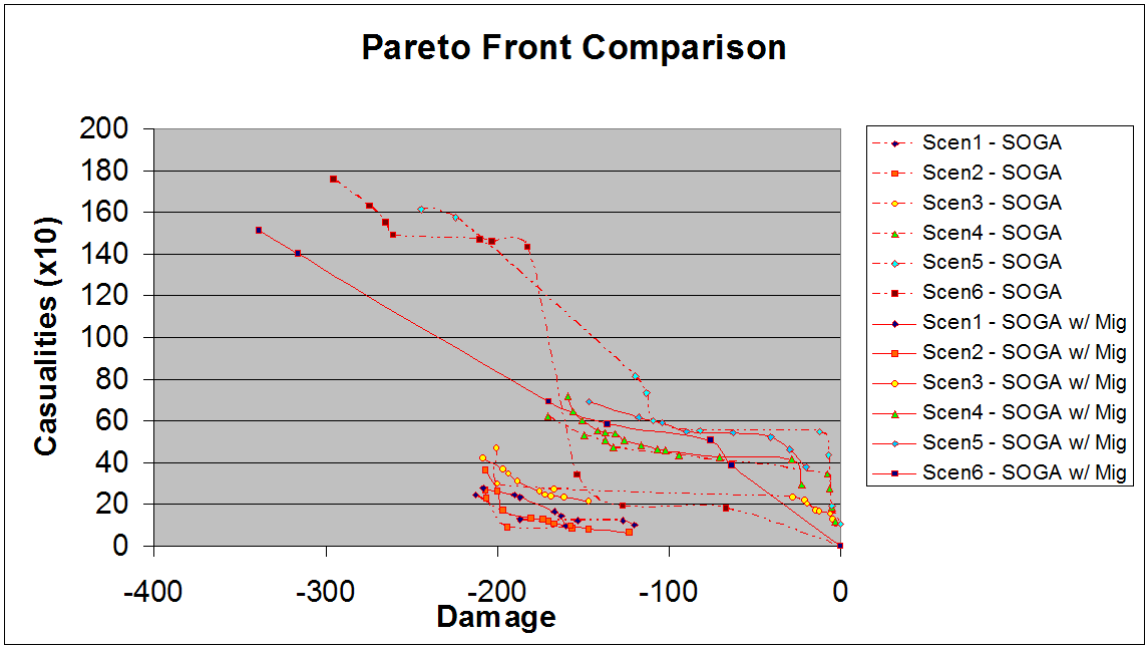


Figure 6.12: This graph compares the $\mathcal{PF}known$ of scenarios. SOGA using the add migration rule is shown with solid lines and SOGA shown with dashed lines. Each scenario level has the same point symbol for both.

| Scenario | SOGA | SOGA w/ Mig |
|---|---|---|
| 1 | 39871.522 | 38915.92 |
| 2 | 39334.54 | 39336.21 |
| 3 | 34729.26 | 36665.88 |
| 4 | 26172.31 | 24528.69 |
| 5 | 22354.73 | 21254.46 |
| 6 | 34132.44 | 34917.4 |

Table 6.5: Hypervolume for Migration Test

The $\epsilon$-Indicator shows the system relatively even through the first several scenarios, however, in Scenario 6 the migration including setup has a superior value. The graph, in Figure 6.12, shows that most of the dominance in the original setup is on the side of minimized causalities. In almost all cases the migration extends towards the ends of the damage range. This makes sense because this implementation of mi-

gration is geared towards getting the agents to the target area, thus large damage and large causalities.

| Scenario | SOGA-W/Mig | W/Mig-SOGA |
|---|---|---|
| 1 | 9.6 | 3.3 |
| 2 | 4 | 0 |
| 3 | 7 | 5 |
| 4 | 9.6 | 5.3 |
| 5 | 97 | 84.3 |
| 6 | 39.3 | 43.3 |

Table 6.6: $\epsilon$-Indicator for Migration Test

Table 6.7 show the Kruskal-Wallis analysis of the optimization rates. The objective of this analysis is to show the difference in the means of each epoch or scenario for the two test sets. This result shows significance in the population distribution beyond just the best solutions found. In 8 of 12 cases the distributions passed the .05 p-value indicating a false null hypothesis. This means that migrations populations are statistically different, and through inspection better, than the previous setup.

| Scenario | Objective Damage | Objective Casualties |
|---|---|---|
| 1 | .1890 | .0659 |
| 2 | .0010 | .0877 |
| 3 | .9476* | .0010 |
| 4 | .0010 | 0010 |
| 5 | .0010 | .0001 |
| 6 | .0010 | .0010 |

Table 6.7: Kruskal-Wallis P-Value on Migration

After visual inspection, statistical analysis migration is shown to improve the performance of the system. Although there are some outliers in the previous setup that are more optimized for minimized causalities, the intended increase in damage emerged.

## *6.3 Bee Attack Structure*

The testing of the Bee Attack algorithm structure is more involved that the previous behavior. The behavior itself is non-trivial so validation in the domain and statistical dominance over the previous configuration are both investigated. Several steps of the behavior are explored specifically. With this behavior set the intended behavior emerges but there are also other unintended emergent behaviors discuss. The statistical analysis of the validated behavior is shown using the previous techniques.

*6.3.1 Behavior Validity.* Validation of each aspect of the Bee-inspired Attack comes from inspection of the visualization. There are three aspects of the behavior: Target Reconnaissance, Target Choice and Threshold based attack. Figure 4.2 shows the behavior performing the first aspect. Table 6.8 shows the before and after animations of a swarm making a common decision on the target selection, after full reconnaissance was accomplished. (This is validated by low level analysis of the state throughout the sequence.) Note in the right diagram the swarm is completing its second pass and in the left diagram each agent has decided to engage.

Finally, if the system finds itself in a position where it does not have enough agents to successfully attack it maintains a stand off range. Figure 6.13 shows two agents (1 UAV, 1 UCAV) who have broken from the swarm as the rest of their subswarm has died. They continue to stay out of range of the targets. This is because the control set thresholds indicate the engagement would not be successful.

Several other emergent behaviors are observed as well. Analysis of individuals in the population has turned the intended attack against itself. For instance, several control sets use a negative weighting on the initial attraction to the target in order to avoid the target more effectively. This forms a passive swarm that favors the lower right of fitness functions in the population. There are also situations were the threshold got so high that the agents went into orbit outside the targets effective engagement radius.

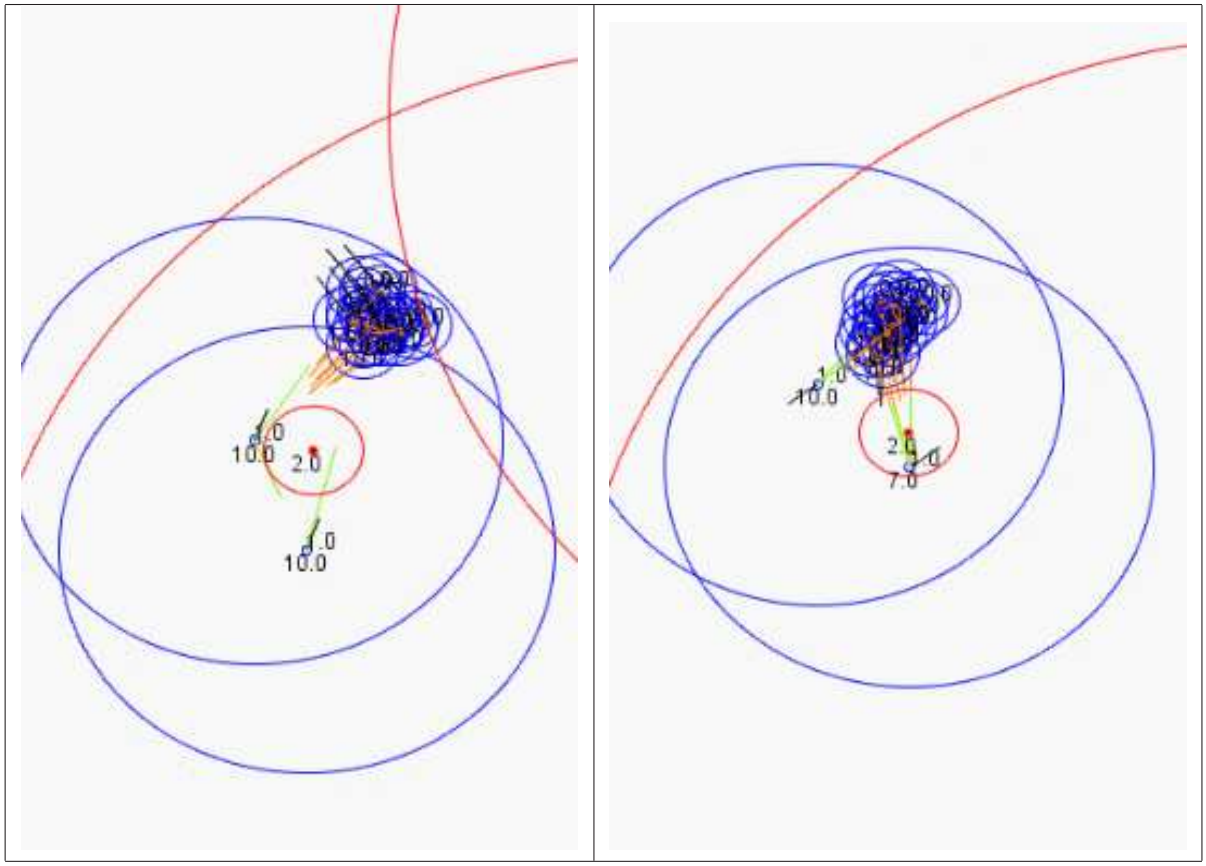


Table 6.8: The swarm before and after target decision.

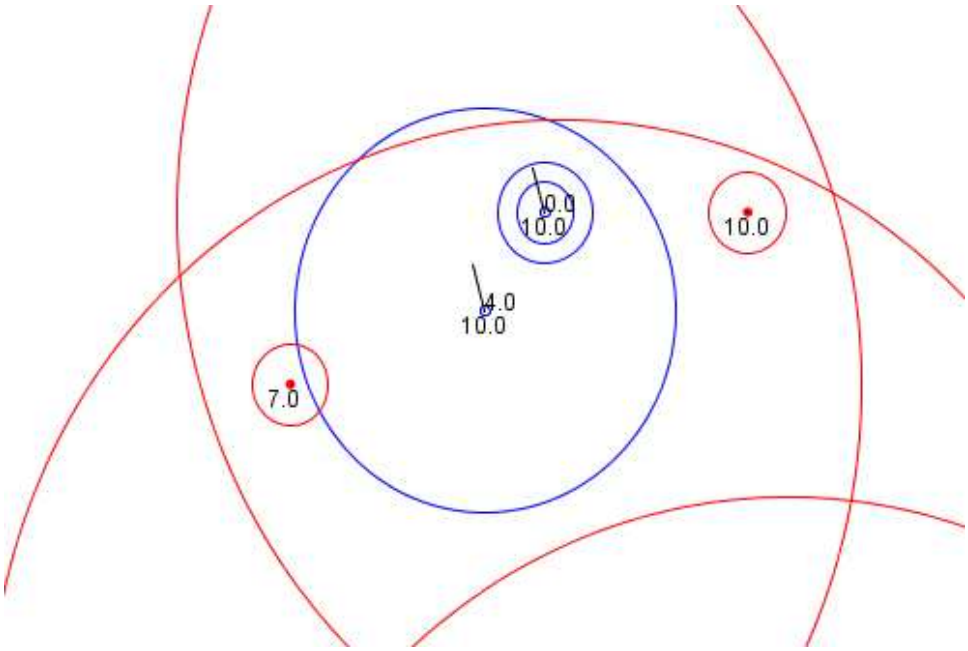


Figure 6.13: A UAV and a UCAV do not have the fire power to attack the targets, as defined by the control threshold. For this reason the are circling the adjacent targets in the set.

Other attack behaviors also emerged. The swarm capitalized on the difference of the control structures between the UAVs and UCAVs (each has their own set of BAs). In one case the UAVs would circle the target on opposite sides in reconnaissance mode and the swarm cohesion drew the UCAVs directly between them and engaged. Several unexpected behaviors like this emerged as the system "optimized" the controls over the solution space.

*6.3.2 Incremental Test.* Several figures and tables show the statistical dominance of the control sets with the Bee-Inspired attack behavior set included. Figure 6.14 shows the population of 30 individuals that are chosen for the next generation in the Bee-Inspired Attack configuration described in Section 5.4. Upon inspection the systems seems to optimize much more congruently and focuses toward the front. (Note the overlapping population for the first scenarios, this did not happen in previous tests. also, Scenario six did not match because it had twice as many targets and agents as the previous 5.)

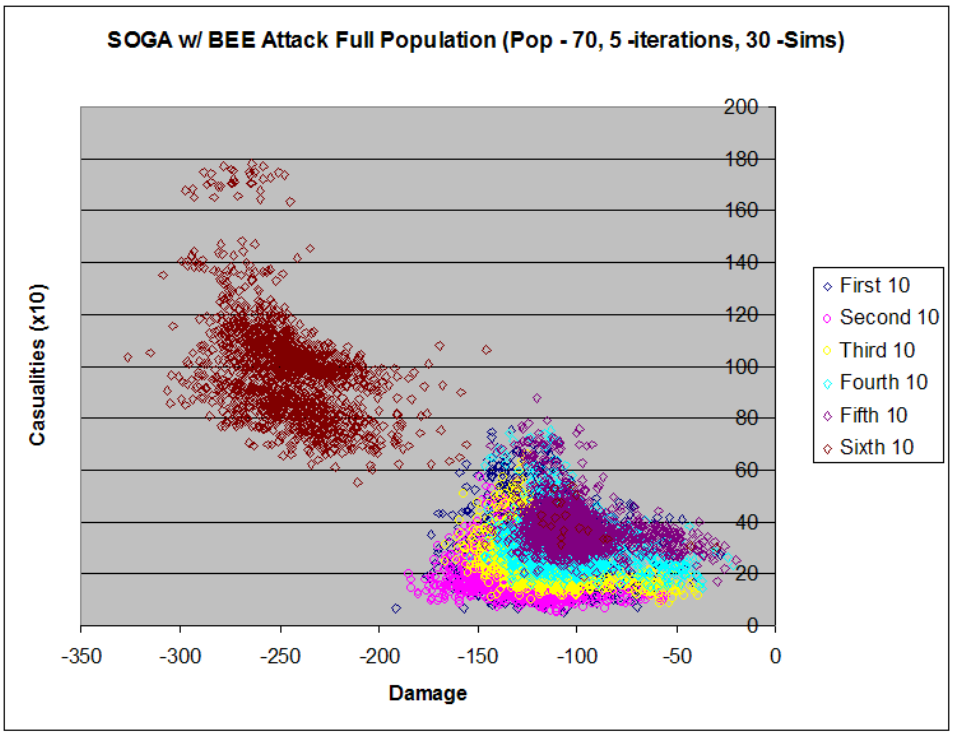


Figure 6.14: In this graph SOGA, using the base rules, Migration and Bee attack, ran 5 iterations of 30 simulations per fitness function over 60 generations with a population of 70 individuals. Six scenario files are used that increased the difficulty of the evaluation.

Figure 6.15 shows the optimization rates of the system over the 60 generations. Two things are worth noting in this graph. First the average damage is about 1/3 better than the previous test and the causalities are slightly better. Second, the variances shown are much smaller even later in the test.

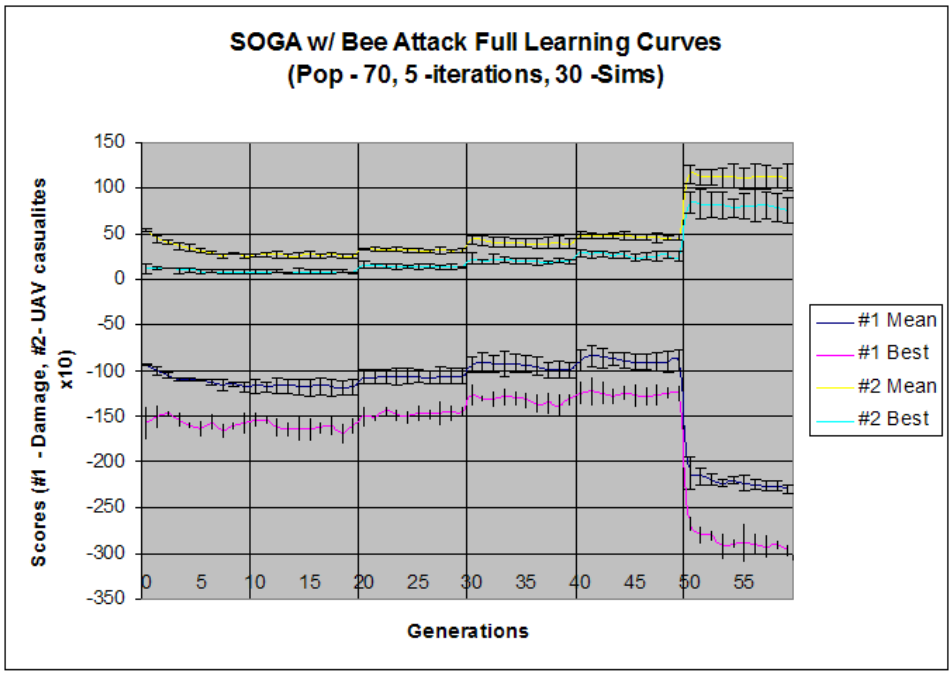


Figure 6.15: The base rules set are used with the Migration and Bee Attack rules added. Mean and Best score by 60 generation with 70 populations for five iterations with 30 simulations for all runs. Vertical lines mark changes in the scheduled scenario. Mean score standard deviation is indicated by the intervals.

Figure 6.16 indicates $\mathcal{PF}known$ of each scenario for the setup with migration and the setup including both migration and Bee-Inspired attack. Their is a noticeable difference in the two sets. In several of the cases the Bee-Inspired attack set completely dominates the previous version.

Table 6.9 confirms what is initially seen in the front comparisons in Figure 6.16. In all but one case the Hypervolumes of the front have superior values to the previous setup. Especially note worthy is the 30+% jump in the final scenario.

The $\epsilon$-Indicator paints the same story. Especially when analyzed in combination with Figure 6.16. All of the error indications for the Bee-Inspired Behavior come

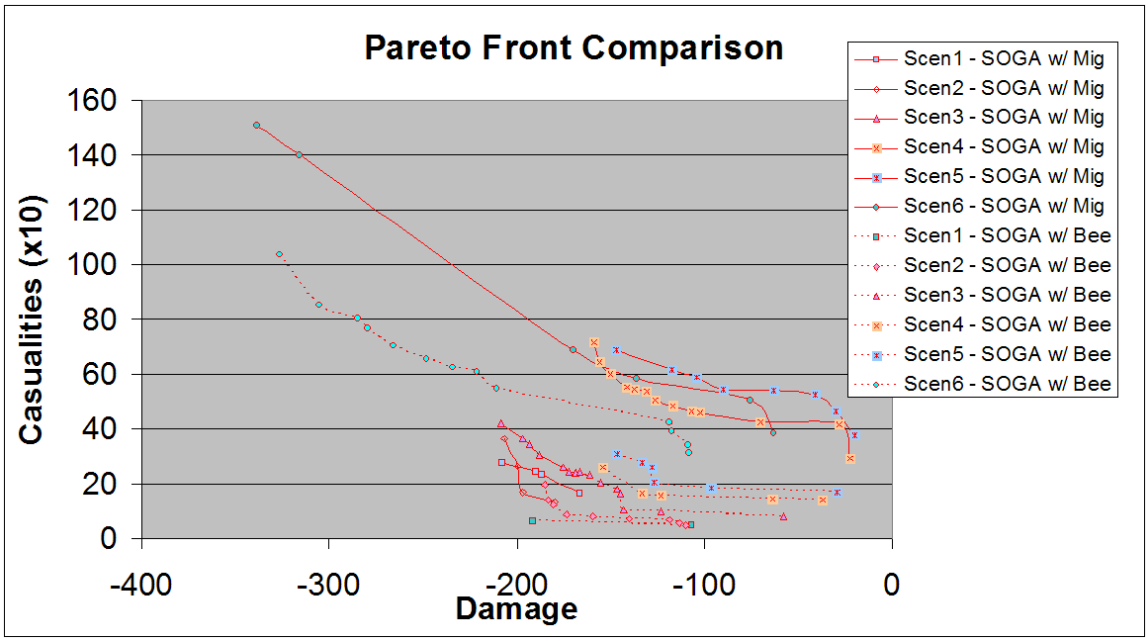


Figure 6.16: This graph compares the $\mathcal{PF}known$ of scenarios. SOGA using the add migration rule is shown with solid lines and SOGA with Bee Attack shown with dashed lines. Each scenario level has the same point symbol for both.

| Scenario | SOGA w/Bee | SOGA w/ Mig |
|---|---|---|
| 1 | 37193.39 | 38915.92 |
| 2 | 35796.68 | 39336.21 |
| 3 | 31480.9 | 36665.88 |
| 4 | 28310.08 | 24528.69 |
| 5 | 26385.31 | 21254.46 |
| 6 | 47407.84 | 34917.4 |

Table 6.9: Hypervolume analysis for Bee Inspired Attack Test

from the movement of the front towards the bottom left. The few sizable errors in the migration column come from heavy damage heavy loss ratios. The Bee-Attack is meant to combat this and therefore tends toward safer attack situations with less casualty to damage ratios.

| Scenario | SOGA W/Bee - W/Mig | W/Mig-W/Bee |
|---|---|---|
| 1 | 10.3 | 15.6 |
| 2 | 70.3 | 8.3 |
| 3 | 97.3 | 31.6 |
| 4 | 38.6 | 4.6 |
| 5 | 23.3 | 0.6 |
| 6 | 56.6 | 12 |

Table 6.10: $\epsilon$-Indicator for Bee Inspired Attack Test

Table 6.11 shows the independence of the populations. In 7 of 12 situations the populations are shown independent by falsification of the null hypothesis on a .05 p-value test. The remaining p-values favor the new attack setup but do not show significance.

| Scenario | Objective Damage | Objective Casualties |
|---|---|---|
| 1 | .2265 | .4306 |
| 2 | .2371 | .0488 |
| 3 | .0010 | .7628 |
| 4 | .3578 | .0010 |
| 5 | .0010 | .0010 |
| 6 | .0010 | .0010 |

Table 6.11: Kruskal-Wallis P-Value with Bee Attack

Analyzing the population produced during the tests with the Bee-Inspired Attack versus the previous test paints the clearest picture. The population is focused around the front. Every indicator and statistical analysis done on these tests confirm this finding. With the intended emergent capability and those unintend, this behavior set has completed objectives 4 & 5 and shown a dramatic increase in effectiveness.

## *6.4 Control and Attack Optimization*

There are two aspects to the discussion of control and attack optimization results. One compares the effectiveness of the controller to the previous work. The other presentation groups all of the testing together to show the successful advancement of the computational UAV simulation system.

*6.4.1 DE-Inspired Controller.* The intent of introducing the DE-inspired controller is to find a more effective and flexible arbitrator than the Neural Netwrok implementation [59], that could operate in a dynamic space. The tests for this first section are described in Section 5.5.

Figure 6.18 shows the population of the system after the introduction of the new controller. Worth of note in this figure is the distribution of the population. In Figure 6.14 the system focused heavily on the middle of the front. In this instance, however, the population covers a broader spectrum of the front.

Once again the trends shown in Figure 6.18 are similar to those in Figure 6.15. The learning curves improve in each epoch. The standard deviations show a similar distribution of the population.

The comparisons of the fronts in Figure 6.19 show that the results with the DE controller are again similar. The population spreads about the fronts in a similar manner. In every epoch the front pushes to the same reachs of the damage objective versus causalities. It appears that the previous testing without the new controller does perform slightly better.

Tables 6.12, 6.13, and 6.14 all indicate the same trend. In the Kruskal-Wallis tests the p-values favor the DE in some cases but mostly are neutral or favor the previous controller.

According to the statistical analysis in Tables 6.12, 6.13, and 6.14, the systems are relatively similar, if not, they favor the NN controller. This is not alarming. Neural Networks are reasonably good, [67] at learning environments with which they are

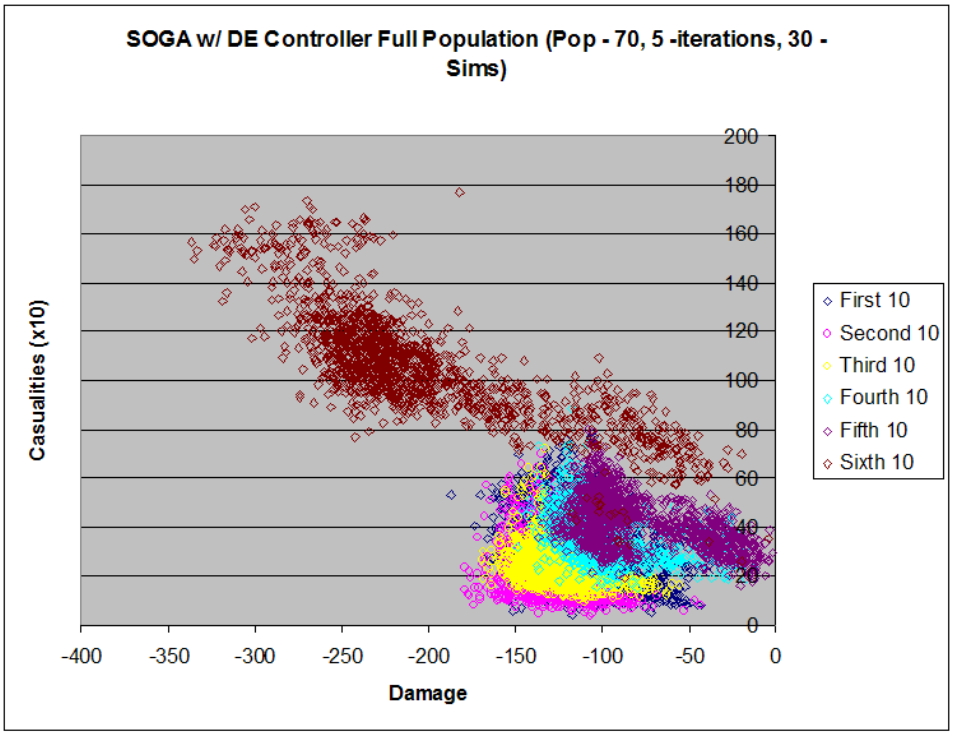


Figure 6.17: In this graph SOGA with all rules and the DE controller, ran 5 iterations of 30 simulations per fitness function over 60 generations with a population of 70 individuals. Six scenario files are used that increased the difficulty of the evaluation.

| Scenario | SOGA w/Bee | SOGA w/ DE |
|---|---|---|
| 1 | 37193.39 | 35392.07 |
| 2 | 35796.68 | 34955.38 |
| 3 | 31480.9 | 31943.61 |
| 4 | 28310.08 | 27198.79 |
| 5 | 26385.31 | 24310.57 |
| 6 | 47407.84 | 42899.3 |

Table 6.12: Hypervolume for DE

| Scenario | SOGA W/Bee - W/DE | W/DE-W/Bee |
|---|---|---|
| 1 | 6 | .6 |
| 2 | 8 | 1.6 |
| 3 | 0 | 8.3 |
| 4 | 5.3 | 0.3 |
| 5 | 4.3 | 0.6 |
| 6 | 25.3 | 0 |

Table 6.13: $\epsilon$-indicator for DE

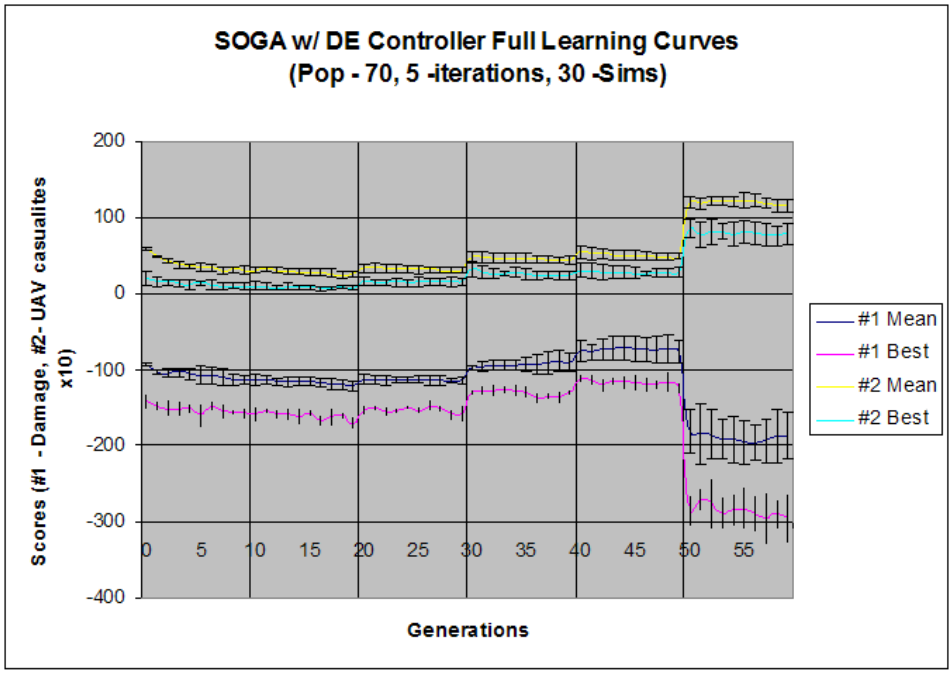


Figure 6.18: All rules and the DE controller. Mean and Best score by 60 generation with 70 populations for five iterations with 30 simulations for all runs. Vertical lines mark changes in the scheduled scenario. Mean score standard deviation is indicated by the intervals.

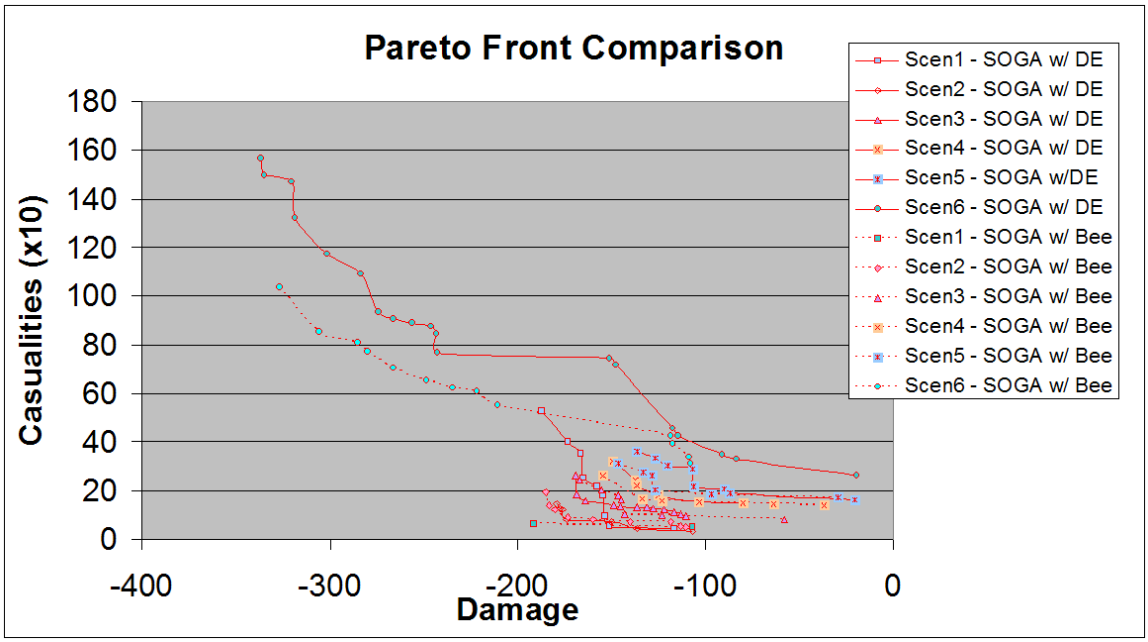


Figure 6.19: This graph compares the $\mathcal{PF}known$ of scenarios. SOGA with Bee Attack shown with dashed lines and the addition of the DE controller shown in solid lines. Each scenario level has the same point symbol for both.

| Scenario | Objective Damage | Objective Casualties |
|---|---|---|
| 1 | .2265* | .4306 |
| 2 | .2371* | .0488 |
| 3 | .0010* | .7628 |
| 4 | .34578 | .0010 |
| 5 | .0010 | .0010 |
| 6 | .0010 | .0010 |

Table 6.14: Kruskal-Wallis P-Value with DE
* - These three p-values favor the new controller setup.

familiar and/or are not dynamic in nature. For this reason the DE-Inpired controller is favored and is used in the next phase of test. The ability of the DE controller to respond to information outside what it has seen is its strength, still performed well.

*6.4.2 Attack Scenarios.* To test the new controller, the attack scenario set with strong target sets is used. The controller allows for easy inclusion of new behavior types. An increase in BAs simply means an increase in the number of foci that control and move about the abstract state space. Addition of more foci increases the controllers ability to map the space more accurately. For this reason, these tests increased the number of BAs. Six BAs are used to give enough foci to cover the expected abstract states based on expert knowledge.

Figure 6.20 appears much different than its predecessors as presented in Figures 6.20 and 6.14. With 9 scenarios being tested, only one of which overlaps with the previous tests, pulling out individual scenario populations proves more difficult. The population near the top in pink squares is the 6th Scenario for the previous tests. The $\mathcal{PF}known$ is illustrated by the green line of circles. In comparison to the previous 6th scenario populations this population covers a large space with a more distributed front. It is also pushed about 100 points in damage, or another full target forward.

The optimization curves in Figure 6.21 show learning throughout the testing. The increase variances come from two factors. First, with more BAs there are more

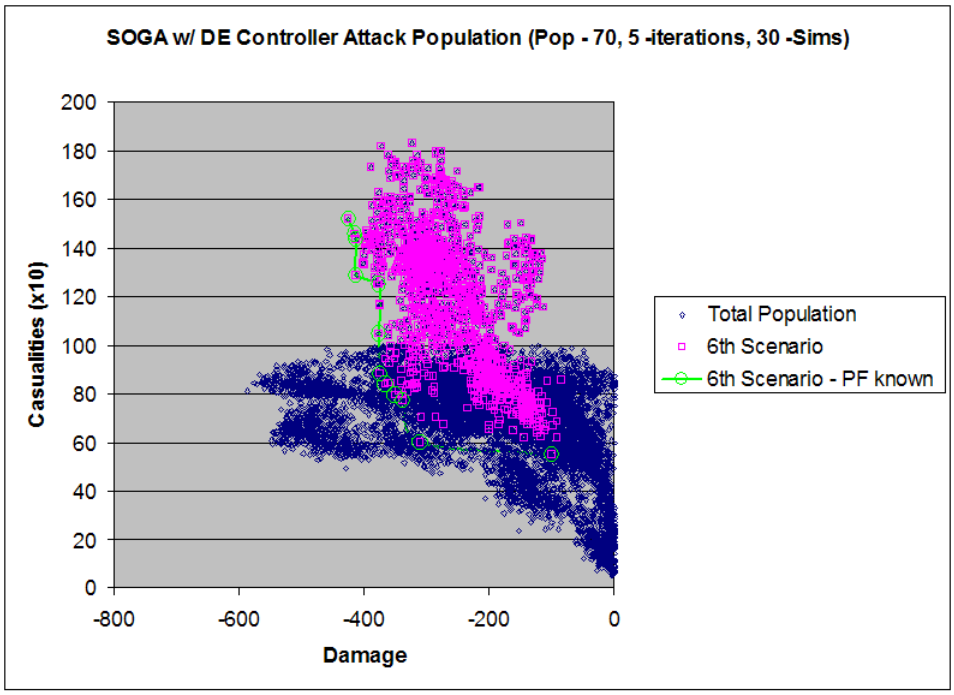


Figure 6.20: In this graph SOGA with all rules and the DE controller, ran 5 iterations of 30 simulations per fitness function over 60 generations with a population of 60 individuals. Nine scenario files were used that increased the difficulty of the evaluation.

ways to accomplish the same tasks. Like nature, with added difficulty comes more entropy. Meaning, weaknesses are exploited by the difficult target sets.

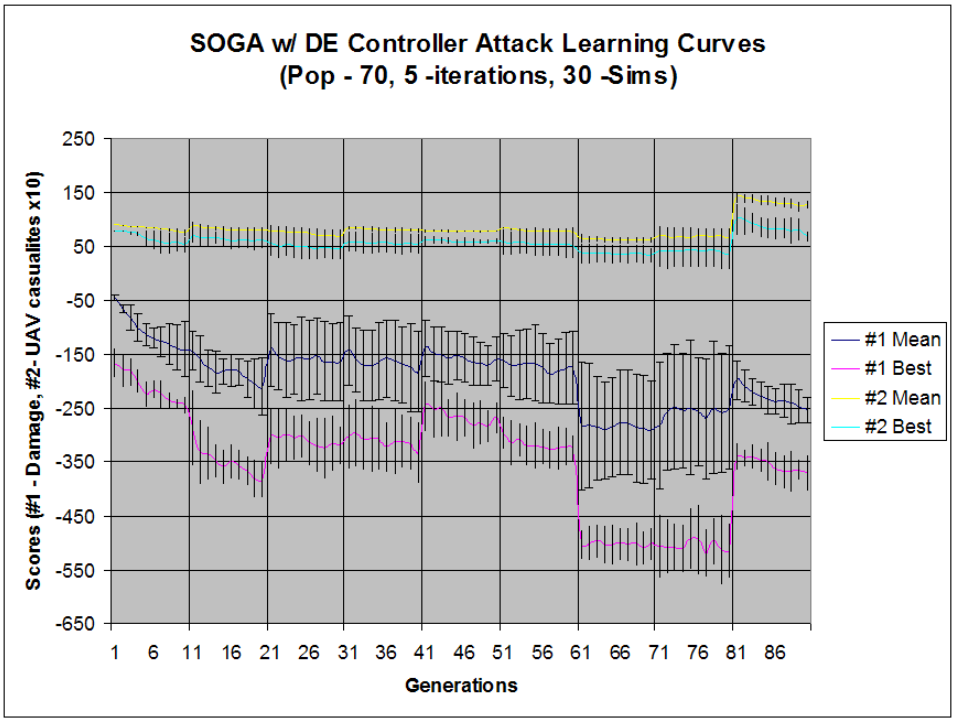


Figure 6.21: Here the all rules and the DE controller. Mean and Best score by 60 generation with 60 populations for five iterations with 30 simulations for all runs. Vertical lines mark changes in the scheduled scenario. Mean score standard deviation is indicated by the intervals.

Figure 6.22 represents the results from the last test and the previous two test's 6th Scenario front side by side. Note the increase in the damage. Also interesting to note is the shape and placement of the curve. Both tests with the new controller create a fuller front. The relative slope of the curves is also interesting; a more aggressive slope indicates that solutions with less causalities inflict more damage. The increase in the casualty to damage ratio is crucial!

Table 6.15 compresses the data seen in the previous tables into one (because there is only one scenario to compare). In all cases the increase number of BAs and testing through the attack scenarios improves on the previous DE-inspire controllers marks. In the bottom right corner the 99.3 is the $\epsilon$-Indicator dominance of the new

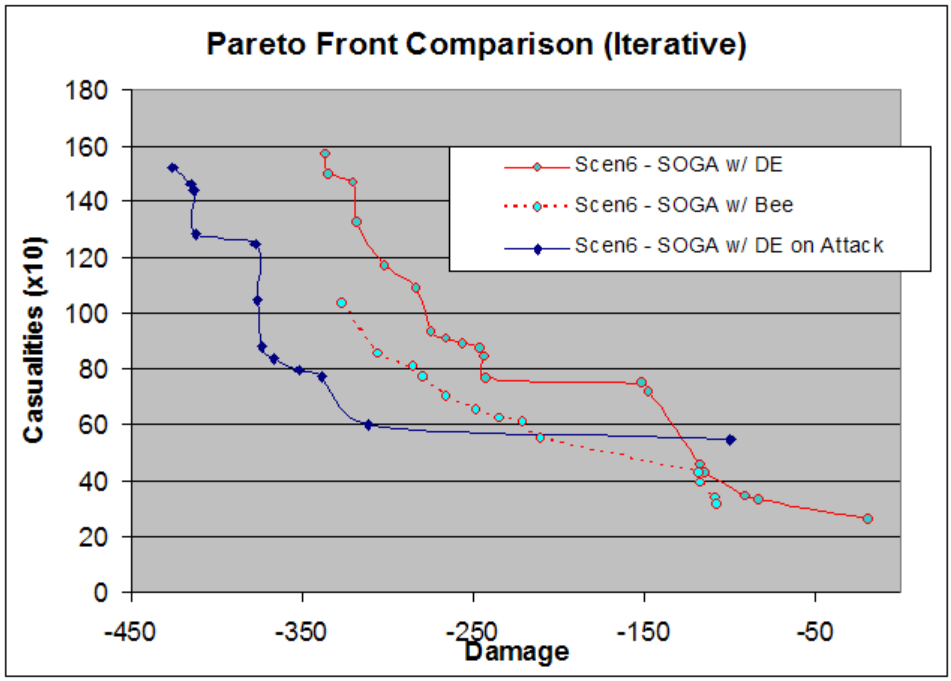


Figure 6.22: This graph compares the $\mathcal{PF}known$ of scenarios.

controller over the old. The value of 14.3 is the opposite, which stems from a more cautious configuration in the bottom right of Figure 6.22.

| Type | Before | After | No Advanced Control |
|---|---|---|---|
| P-value (obj 1) | - | 0.1484 (.0010 best) | - |
| P-value (obj 2) | - | .0010 - (.0761 best) | - |
| Hypervolume | 42899.3 | 55061.76 | 47407.84 |
| Epsilon | 18.3 | 48.3 | 99.3 (14.3) |

Table 6.15: Statistical Comparison of Advanced Setup on Two Scenario Sets

Figure 6.23 compares the final statistics of the system with the original. The blue diamond line to the far left shows the growth. The achieved successes in attack are notable. The top damage number moved from 280 (or just shy of 3 targets) to 430 (or solidly over 4 targets of 6). [59] showed approximately 2/3 success in target kill as well, but the system did not consider casuality rates. Upon visual inspection the difference is notable. Agents rarely colloid or fly out of the target area with the MOEA implementation and improved controls.

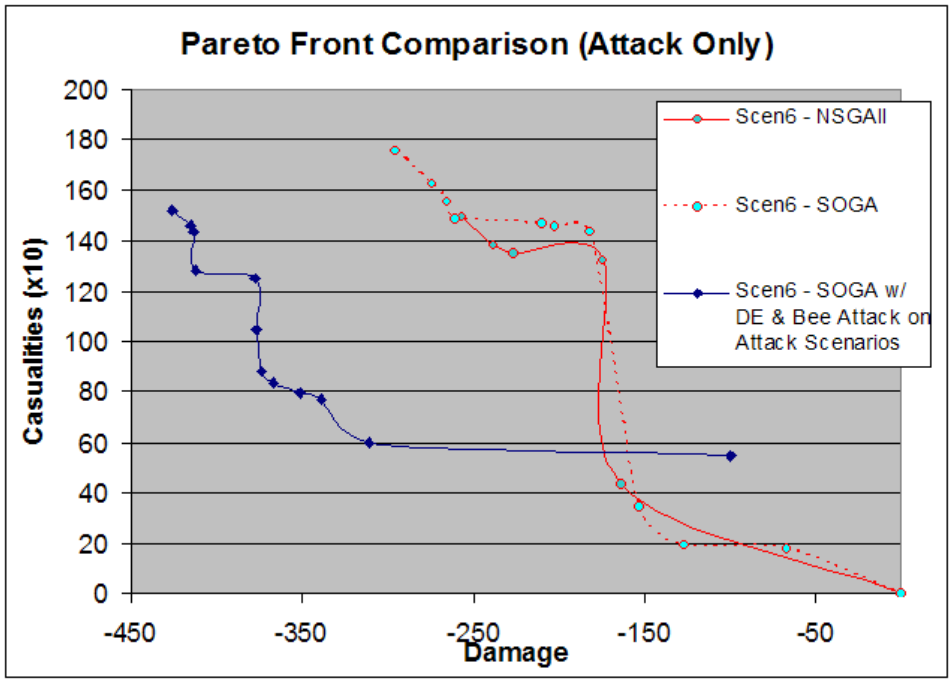


Figure 6.23: This graph compares the $\mathcal{PF}known$ of scenarios.

Table 6.16 shows the statistical analysis of the fronts and populations. The p-values leave no room for doubt in the independence of populations! The hypervolume increase is approximately 70% as well. The final front also dominates the previous two fronts with an error of over 100 points. The 54.6 error indicator in the final front represents the previous tests ability to find a solution that accomplishes nothing without causalities.

| Type | NSGA-II | SOGA | Advanced Control & Behavior |
|---|---|---|---|
| P-value (obj 1) | 0.0002 | 0.0002 | - |
| P-value (obj 2) | 0.0002 | 0.0003 | - |
| Hypervolume | 31301.71 | 34132.44 | 55061.76 |
| Epsilon | 130.6 | 170 | 54.6 (both) |

Table 6.16: Statistical Comparison of Original and Advanced Setups

This section discusses the results associated with Objectives 6, 7, and 8. Although statistical dominance is not achieved in Objective 7 this is not a problem. The

flexibility of the new controller is evident in the increased performance when the BAs numbers were changed. Overall the test showed that this step in the iterative process of optimized effectiveness workedas desired.

## *6.5 Chapter Summary*

The results and analysis of the research effort follow the design of experiments as discussed in Chapter V. Table 6.1 presents the testing schedule. Section 6.4.2, in this chapter outlines the final results, indicating marked gains, increased damage by 75% and decreased casualties by 15%. The single objective model failed to provide acceptable and understandable results. Overall the testing sequence accomplished its objective. It showed very successful results in solution optimization.

# VII. Conclusions

This summarizes the integration of the problem statement, the objectives outlined to address the problem, recaps the successes of the investigation, and suggests future research in continuing this very successful investigation.

The results from this research generated significant strides in the area of autonomous UAV swarm control:

- The development of a **new abstract UAV swarm control** model that flows from a Markov model, the POMDP. Using this innovative model, a **new problem domain iterative decomposition technique** evolves the required UAV agent rule sets.
- With this decomposition, single-level UAV flight behaviors and **new multi-level bio-inspired attack behaviors** are instantiated. Thus with evolutionary computation, a generalized **entangled computational cooperative control hierarchy** is constructed providing the desired emergent combined UAV behaviors. This evolved architecture abstraction is a **paradigm shift** in cooperative autonomous UAV control.
- Validation of this innovative approach through the continuing development of an **extensive software simulation, animation and statistical analysis** indicates that such a self-organized autonomous swarm of UAV agents can performed desired operations in a dynamic environment.

The following sections briefly discuss these contributions through the research objectives.

## *7.1 Develop Model*

The first objective outlined in Section 1.2.1 requires a model that could describe the abstract UAV swarm problem domain. After considerable searching and evaluation, the Markov Model, POMDP, is selected as discussed in Section 3.1.2. Although not a perfect Model, POMDP is chosen because in our problem domain the entire

state space is not known and the affects of the transition actions are not known. Thus, the tuple $D(S, A, T, O, R)$ serves as the basis for developing the whole state model (see Section 3.1.2).

With this abstraction, a non-deterministic approach using Biologically Inspired Self Organizing systems is developed. A new decomposition method allows desired behaviors to emerge. This new technique is called the 'U'-decomposition. It focuses on decomposing the problem domain into basic pieces that could be handled by basic rule sets. The structure then ties the rules sets in an entangled hierarchy, from which the UAV swarm behavior emerges. This does take careful crafting. When accomplished correctly creates a system that evolves ways to get things done dynamically, instead of through an explicit a priori instantiation. As such it represents a very limited view of the human mind's entangled dynamic control hierarchies which attempts to generate efficient and effective problem solutions. For this research the modeling technique focuses on UAV swarm implementation in a computation target engagement simulation environment. The design process initially decomposes the problem objectives into two sub objectives, getting to the target and optimizing the target engagement. The first is relatively easily implemented through a behavior called migration. The second needed further decomposition which results in a three step process: target reconnaissance, target analysis, and threshold based target engagement. All three are accomplished autonomously and the aggregation of the agents in the swarm reflects the desired behavior effect (see Section 3.1.2). Overall this approach and decomposition technique provides very fruitful results. The rules are simple, easily implementable, and work in a dynamic environment. It relies on non-deterministic aspects and therefore has its disadvantages in human understanding of the explicit underlying processing, however, this is acceptable in such a convoluted computational domain.

### *7.2 Integrate Attack Behaviors and Entangled Hierarchy*

The next objective (Section 1.2.1) is to integrate the new behavior constructs into the existing swarm simulation and animation [59]. Software engineering principles are not only applied in order to allow for replacement and modification of independent pieces of the software, but the entire software simulation package is restructured for ease of modification. Once accomplished the addition of a Self Organizing Genetic Algorithm facilitated testing of all further additions without detailed consideration with parameter values. The behavior sets are those of the original UAV package [59] in order to decouple the behaviors, controller and UAV. The addition of the migration behavior fits easily into the new construct. The advanced attack behavior (Bee-Inspired Attack) required definition of a sub-controller based on abstracted state. This sub-controller integrated several other behaviors into the advanced attack; a multi-level behavior formulation. This multi-level behavior modeling approach should generate more complex entangled cooperative control hierarchies that reflect not only effective performance but efficient operation.

As a result of the complexity of the emerging structure from the advanced attack behaviors a new controller is need to be implemented. The new controller (arbiter) is required to operate in a dynamic environment and over come the current artificial Neural Network's in-ability to understand and operate on information not previously available. With this implementation an increase in BA and sensor information is also possible. The result of this integration formed a software package that is decoupled, reusable, and well engineered. Separate packages for the GA, Controller, Agents, and Behaviors make it very usable for continued development. The added controller and behavior sets are also very effective at "optimizing" target engagement.

This modeling approach is critical not only to autonomous self-organized swarm structuring, but to the abstract analysis of any agent system behavior. With evolutionary computation, a generalized entangled computational cooperative control hierarchy has been constructed that can provide the desired emergent combined agent

behaviors. This evolved architecture abstraction is a paradigm shift in cooperative autonomous agent control.

### *7.3 Validate Model*

The final objective of this research is to validate the created model through statistical analysis and evaluation (Section 1.2.1). This is accomplished through an iterative process whereby each test is built on the results of the last. Three types of information are analyzed: optimization rates, population characteristics, and visualization through simulation. The initial aspect of the testing and validation is accomplished with the new algorithm, the SOGA. After comparing it to the known single objective and a Monte Carlo Simulation, it is tested against a known MOEA, NSGA-II. SOGA outperforms the first two, and is not statistical significant, according to a Kruskal-Wallis test, with the population from NSGA-II (Section 6.1.3). The design intent is to create a GA that could run in the computational environment and not require constant parameter tuning. The testing of this implementation validates that intent. Several behaviors are tested against the original behavior set from [54] using the new SOGA. In both cases, migration and advanced attack, the system statistically outperformed its predecessor in both damage and casualties. Comparison with the original behavior set as evolved by SOGA showed PFknown dominance, marking the new behaviors as valid as well (Section 6.1.3).

The system "optimizes" the target engagement environment through several steps. Addition of a new controller, the DE-Inspired controller, is only as good as the previous implementation, in the benchmark tests. However, the ability for the new controller to operate in more dynamic environment makes it much more attractive.

Tested with the new attack scenarios. the increased number of BAs the controller really took flight. Analysis between the previous tests and attack optimized tests shows a significant difference, statistically, in the population (Section 6.4.2). Finally, comparison of the last set of data obtained with all the new elements and the specific attack scenarios showed the most remarkable results. There is a 50%

increase in effective damage between the solutions found in the original configuration, using NSGA-II or SOGA, and final configuration. The average damage on the last generation has increased by 75% while the causalities are decreased by 15%. Overall capabilities of the SO designed swarms has improved dramatically through increased ability to inflict damage with lowered casuality rate.

## *7.4 Future Work*

The results of this research is of course not currently deployment capable for physical low-cost effective UAVs. Setting aside the lack of detailed sensor and communication capabilities, the research thus far has explored only some aspects of self-organized (SO) control. The swarm however has dynamic swarm formation, waypoint navigation, and approaches to successfully engaging a target. But this is not the full set of tools needed to feasibly be deployable. Thus, future efforts should include a variety of developments:

*Generally*

- Include a broader set of successful, but different role capabilities in the form of bio-inspired and other behavior sets in Swarmware.
- Include more distributed computation and data storage techniques, for target recognition and distributed processing (efficiency) - Extended testing to exploit increased efficiency.
- Apply our UAV modeling principles, decomposition and design constructs to other POMDP problem domains (cybercraft, robo-insects, and other hardware and software agent systems).
- Movement of the modelling principles to other domains (cybercraft, robo-insects, and other hardware and software agent systems).
- Continue to develop the entangled architecture with multi-level behaviors structures.

- Development of internal system subcomponents for any computerized system.
- Continuing evolution of SO based 'U'-Decomposition principles.
- Add to simulations, precise models of sensors and communication protocols as well as dynamic cooperative control equations.
- From the model and Swarmware simulation results, apply the bio-inspired cooperative control concepts to the AFIT Advanced Navigation Technology (ANT) Center small aircraft and flight test.

  *Specifically*

The development of entangled cooperative control techniques is in its infancy and has considerable development before its full capabilities would be revealed. UAV swarms models and the current implementation improvements can be made:

- Add movement in 3D space; 6 degrees of freedom dynamic model ($>$ 25cm, Macro-level $<$ 10cm (insects?), micro-level; DARPA micro UAVs $<$ 1cm).
- In addition to the target damage and UAV casuality multiobjectives include UAV energy minimization and probability of target damage.
- Apply Swarmware to the AFIT 3D UAV Path Planning, Scheduling and animation simulation.
- Enhance visualization tools that better model in 3D (Open Dynamics Engine).
- Refine other attack bio-inspired strategies such as found in other insects, animals and birds of prey and integrate behavior into 2D Swarmfare and 3D simulator.
- Behavior optimization in other sub-problem domains or roles (CAS, EW, ISR, CAP...).
- With our current UAV emergent behavior of formation control, target reconnaissance, target analysis, and threshold based target engagement, target attack, obstacle and collision avoidance, include target movement and tracking and possibility target recognition techniques.

With the inclusion of other UAV behaviors, testing would be more extensive providing a better platform for UAV cooperative control analysis. A similar set of behaviors could be trained on each desired role and then the Behavior Archetypes (BAs) could be combined. It would also be possible to have dynamic behavior sets defined at runtime. (Although the GA should negate any unneeded behaviors currently it would allow the user a clearer understanding.)

Most importantly any future research should continue to extend the ideas of Self-Organization decomposition. Remaining unchained to the more deterministic, discrete layered approach is critical to system performance.

Research on effectively abstracting the ideas of emergent SO structure, abstract state description, and minimization of information complexity. Such a construct requires each sub-struct to be treated as an agent and taking in account the information complexity needed to communicate state transitions in the emergent control structures. The development of such techniques is in its infancy and has a lot of grow pains before its full capabilities should reveal themselves.

### *7.5 Summary*

This research investigation set out to improve on the contemporary body of knowledge on autonomous control of UAV swarming vehicles. An innovative self-organized autonomous swarm model of UAVs is developed. The computational implementation of this bio-inspired model performs desired operations in a dynamic environment. The specific implementation in the Swarmfare simulation system has increased dramatically in effectiveness in the target engagement problem domain. More importantly the research effort encompasses generic methods to decompose a problem domain that exploits the SO design employing an efficient and effective entangled hierarchy. This way of approaching the autonomous control should not constrain a designer's ideas. Therefore, more dynamic situations and environments can be skillfully navigated by the SO agents without explicit instantiation. In summary, this research effort is just one of the initial steps that could lead to a more effective

implementation of swarms that one day would become a formidable replacement for man-in-the-loop systems.

# *Appendix A. SO Abstract Model Types*

| Abstract Model Types (AMT) | | | | | | | |
|---|---|---|---|---|---|---|---|
| **AMT** | **Species / Sub-species** | **Operator** | **Parameter / Sensor** | **Condition / Knowledge Base** | **AF Uses** | **Rules** | **Remarks** |
| Path Solver | Molds | Chemical induced movement | cAMP sensor | internal and external cAMP levels | Movement | pg 104-105 (1) | |
| Massing | Bark Beetle | Pheromone production, autonomous movement (larvae Density) | pheromone (chemical) | Pheromone concentration/gradient | Massing | dispersion, time, distance based pg 132 (1) | possible mass of attacks applications |
| Synchronization | Firefly | light, locomotion, timing mechanism | visual | pulse time constant | Communication | undefined pg 151-155 (1) | synchronization of timing sources or mass of attacks applications |
| Construction | Termites | retrieval & placement dirt, path follow | pheromone (chemical) | queen attributes | unknown | pg 392 (1) | |
| Construction | Wasp | Nest construction | touch/visual | Foundational structure, # adjacent cell walls | unknown | pg 430-432 | |
| Dominance / Hierarchy | Wasp | Challenge | touch | Force, Rank | Structuring | 460-461 (1) (Tsu) | |
| Foraging | Bee | Dance, Foraging | visual | dance rhythm | IR | based on number of foragers and dance length 207-208 (1) | transfer of data and resource re-supply point |
| Cluster control | Bee Swarm | Thermoregulation | heat | temperature range, radius, density | unknown | separation and movement inward/outward 294-297 (1) | |
| Classification | Bee Hive | Honeycomb fill structure | visual | oviposition, fill rates, depletion rates | Classification | pg 331 (1) | |
| Foraging | Ant | foraging, marking, feeding, path following | pheromone (chemical) | chemical production, pheromone interpretation | path to target | number of travelers, length, chemical deposit strength, time 229,232-234,239-241 (1) | |
| Offensive Mass | Ant Raiding | foraging, marking, Carrying prey, path following | pheromone (chemical) | crowding, pheromone product/interpretation | Military Mass | pg 269-274 (1) | Multiple speeds, raid size OM: 100Ks |
| Construction | Ant Nest | carrying, pushing, deposit | Touch (Pheromone 2nd) | structure resistance, brood size/location | Building or classifying | Pg 356-360(1) | distribution of pheromones around brood |
| Path Solver | Ant | locomotion, path laying, path following | pheromone (chemical) | Pheromone concentration/gradient | unknown | pg 1 (2) | Leafcutter |
| Classification | Ant Nest | Brood and corpse | pickup, drop | adjacent items, threshold | unk | Pg 152(2) | |
| Construction | Ant Weaver | chaining, weaving | touch | larva silk weaving, agent bridge construction | unknown | pg 1 (2) | |
| Schooling | Fish | locomotion, predator, schooling | visual, lateral line | Adjacent agent information | defense/ swarming | proximity, repulsion, matching and search; pg 180-181 (1) | Swarming, obstacle gradients |
| morphology | Ant | task distribution has a high plasticity | varied | agent based | UAV swarms | sub species allow for more specific task application | Multiple species (minor or minims, medium workers, sub-majors, major) |

Figure A.1: SO Abstract Model Type Table

# *Appendix B. Simulators Comparision*

| UAV Simulators | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Multiple UAV Sims | | | | UAV Control Sims | | | | SWARM Sims | | | |
| **Name** | **MATLAB Multi UAV** | **MultiUAV** | **SwarmFare** | **Simulation** | **UAV Simulator** | **MVCS** | **MUSE** | **RMUS** | **SWEEP** | **SWARM** | **MASON** | **REPAST** |
| **Maker** | MATLAB | AFRL | Ian Price (AFIT) | IcoSystems | Team Australia | Raytheon | MetaVR | Australian Centre for Field Robotics | Case Western | Sante Fe Inst | GMU | SourceForge |
| **Platform** | Matlab (MS) | Matlab (MS) | Java (MS) | MS | unk | Java (MS) | unk | SimComp iler (UAV) | Java | C (Java capable) | Java | Java |
| **Focus** | UAV Swarm Control | UAV Swarm Control | UAV Swarm Control | Swarm Simulation | UAV training, research, and ops | Multi-Disimilar UAV control | UAV training | UAV experiments | Generic SO | Generic SO | Generic SO | Generic SO |
| **Swarm Control Capability** | Primary purpose | Capable | Primary Purpose | unk | none | Unlikely | N/A | exists | Capable | Capable | Capable | Capable |
| **Modularity** | highly | highly | Moderate | unk | unk | unk | Distributed | High | | | Highly | |
| **Existing Swarm Attributes** | * | Cohesion | Cohesion (search and Destroy) | unk | none | none | N/A | some | no inherent UAV | no inherent UAV | no inherent UAV | no inherent UAV |
| **Attainable** | Internet Download (not | Received | Local | unk | FMS | Gov contract | Gov contract | FMS | OpenSource | OpenSource | OpenSource | OpenSource |
| **Support** | Email | Local Good | Moderate | | overseas | email | email | overseas | | wikiSwarm | wikiSwarm | wikiSwarm |
| **Analysis Capablities** | A | H | A | A | A | N/A | N/A | A | N/A | H | A | H |
| **Visualization** | unk | A (2D) | A (2D) | H (2D) | A (3D) | A(3d) | H (3D) | H(3D) | A (2D) | H (2D or 3D) | M (2D) | A (2D) |
| **Enviroment** | unk | V, T | V, T | V, T | V | V | E, V | V, T, E | V | V | V, T | V |
| **Agent** | S, C | S, C | S, C | | S | S | S | S, C | S | S | S | S |
| | | | | | | | | | | | | |
| **Notes** | Needs MATLAB Plugin License | | Working | No Long in use by AFRL | | | | | | | | |

H - High, A - average, L - Low
V - Vehicle, T - Target, E -Terrain
S - Sensor, C - Communications

Figure B.1: UAV Simulator Comparison [6, 31, 38, 46, 48, 59–61, 81]

| Executable | operating system | open source | documented | source code | software design | graphical interface | command line mode | built-in statistical analysis | data monitoring/custom analysis | built-in visualization tools | allocate tasks | supports message passing | parallel platform | scalable | Optimistic vs. conservative | state-saving support |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GloMoSIM | both | y | y | C | FD | - | - | - | - | - | y | y | y | y | | u |
| HLA | - | - | y | - | - | - | - | - | - | - | - | y | y | - | | - |
| SPEEDES | both | l | y | C++ | OO | - | y | - | - | - | y | y | y | - | O | y |
| PARASOL | both | b | y | C++ | u | u | u | y | y | u | y | y | y | u | O | y |
| JSDESlib | both | u | l | Java | OO | - | - | - | - | - | y | y | y | l | u | y |
| DSIM | both | y | l | C++ | u | u | y | - | - | - | - | y | y | - | O | - |
| SPaDES | both | y | y | Java | OO | - | - | - | y | - | y | y | y | y | O | y |
| | | | | | | | | | | | | | | | | |

Win- Windows Platforms; Java- Java Virtual Machine Plaftorms; Lin: Linux Platforms
l: supported but limited; b: beta stage; y: supported in full
u: unknown
-: not supported
OO: Object Oriented; FD: functional decomposition
Both: Windows and Linux

Figure B.2: Parallel Discrete Event Simulator Comparison [22]

Excerpt from Price [59]

## *C.1 Rule Equations*

There are ten different rules governing the way a UAV moves. Each of these rules is mathematically defined in the following subsections and depicted graphically.

*C.1.1 Rule 1: Alignment.* A particular UAV tries to match directions for its velocity with all other UAVs. This is expressed in the following definition where $U_{R_1}$ is the value of rule 1 with respect to $U$. This rule is essentially the same as that used by Reynolds [62].

$$U_{R_1} = \frac{\sum_{i=0}^{|N|} N_i.D}{|N|} \tag{C.1}$$

Other examined forms for this rule include a distance weighted alignment as shown in Equation (C.2). The unweighted version was determined to be less computationally intensive since it required less divisions while resulting in similar behaviors.

$$U_{R_1} = \frac{\sum_{i=0}^{|N|} \frac{N_i.D}{dist(U.P, N_i.P)}}{|N|} \tag{C.2}$$

The the unweighted alignment behavior can be seen in figure (C.1). Basically, this particular rule causes UAVs that can see each other to fly in the same direction. This behavior is effective for making formations fly in the same direction.

*C.1.2 Rule 2: Target Orbit.* This rule provides a behavior causing UAVs to circle around a target at a safe distance. This is performed by first calculating directions that run perpendicular to the line between $U$ and the target. Then, the perpendicular direction that is closest to $U$'s current direction is selected. Determina-

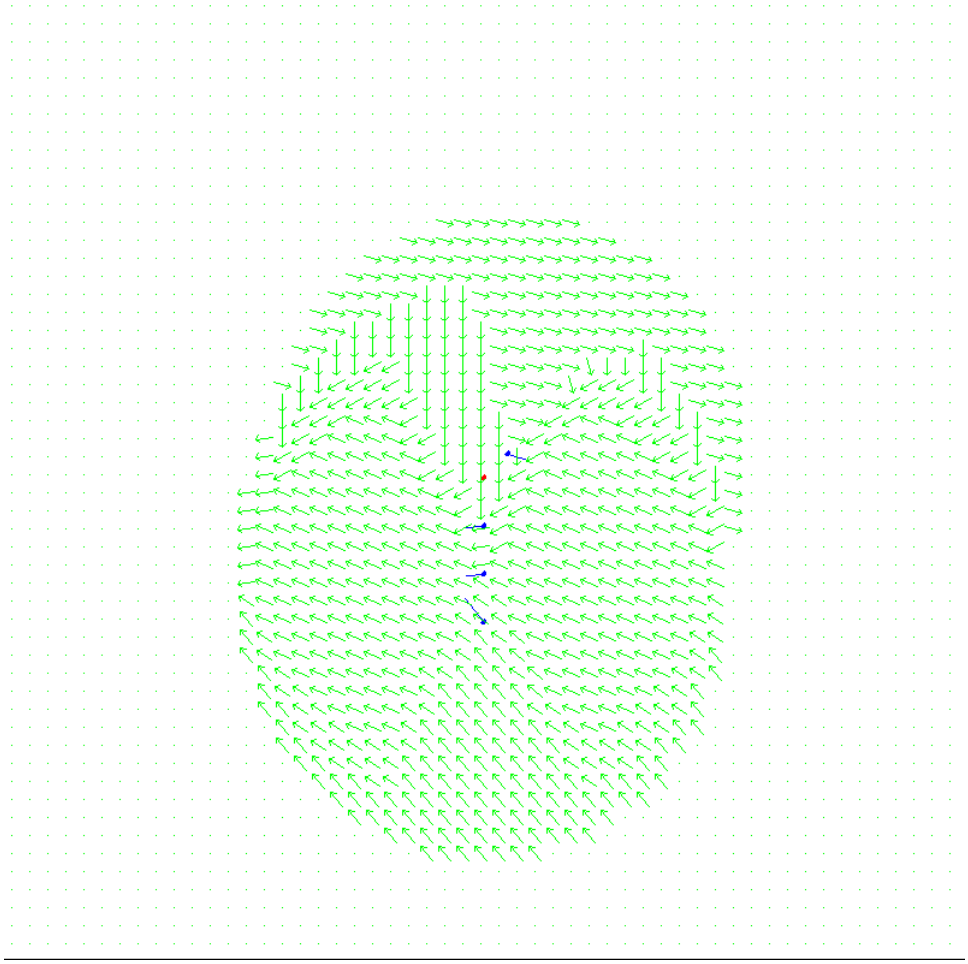

Figure C.1: Field plot for alignment rule. UAVs at (420, 380), (400, 440), (400, 480), (400, 520) with individual velocities indicated by their direction line. Plot assumes a sensor radius of 30km

tion of perpendiculars is performed for each target that $U$ sees. The resulting selected perpendiculars are summed for each target that $U$ is more than 70% sensor range distant. The reason for the effective range for orbits is simple: if a UAV gets too close to a target and engaged, it might as well attack that target rather than simply circle around it. This rule is inspired by Lua [47].

The perpendicular bearings are determined by the following Equations (C.3) and (C.4):

$$d_1(U.P, t.P) = (t.P_y - U.P_y, U.P_x - t.P_x) \tag{C.3}$$

$$d_2(U.P, t.P) = (U.P_y - t.P_y, t.P_x - U.P_x) \tag{C.4}$$

Once the perpendiculars are calculated, the particular one closest to the current velocity is selected. This particular selection is performed in Equation (C.5).

$$Orbit(U.P, U.D, t.P) = \begin{bmatrix} d_1(U.P, t.P) & dist(d_1(U.P, t.P), U.D) < dist(d_2(U.P, t.P), U.D) \\ d_2(U.P, t.P) & otherwise \end{bmatrix} \tag{C.5}$$

The preferred orbiting directions for each known target are then summed for each target that is more than 70% distant. This is accomplished in Equation (C.6). The reason this rule is applicable at a 70% distance is to facilitate cooperative function with behavior rules that cause flat target attraction and flat target repulsion. When these rules are combined, they can cause the UAVs to enter into stable orbits around a particular target. This combination of rules can be seen in Figure (C.2). When combined, these rules cause UAVs to

$$U_{R_2} = \sum_{i=0}^{|\hat{T}|} \begin{bmatrix} Orbit(U.P, U.D, t.P_i) & dist(U.P, t.P_i) \geq .7U.Sr \\ \{0, 0\} & otherwise \end{bmatrix} \tag{C.6}$$

The results of this rule can be seen in figure (C.3). Clearly, when examining Figure (C.3), this rule causes a UAV to prefer to orbit around a target at a safe distance.

*C.1.3 Rule 3: Cohesion.* UAVs are attracted towards each other if the distance between them is greater than a certain range. The influence of attraction towards each UAV is based upon the distance UAV $U$ is from a specified percentage of $U$'s sensor value, $r_1$. This rule is inspired by both Reynolds [62] and Kadrovich [40]. Additionally, this particular version has shown usefulness in previous work [59]. Equation (C.7) demonstrates how this rule is computed.

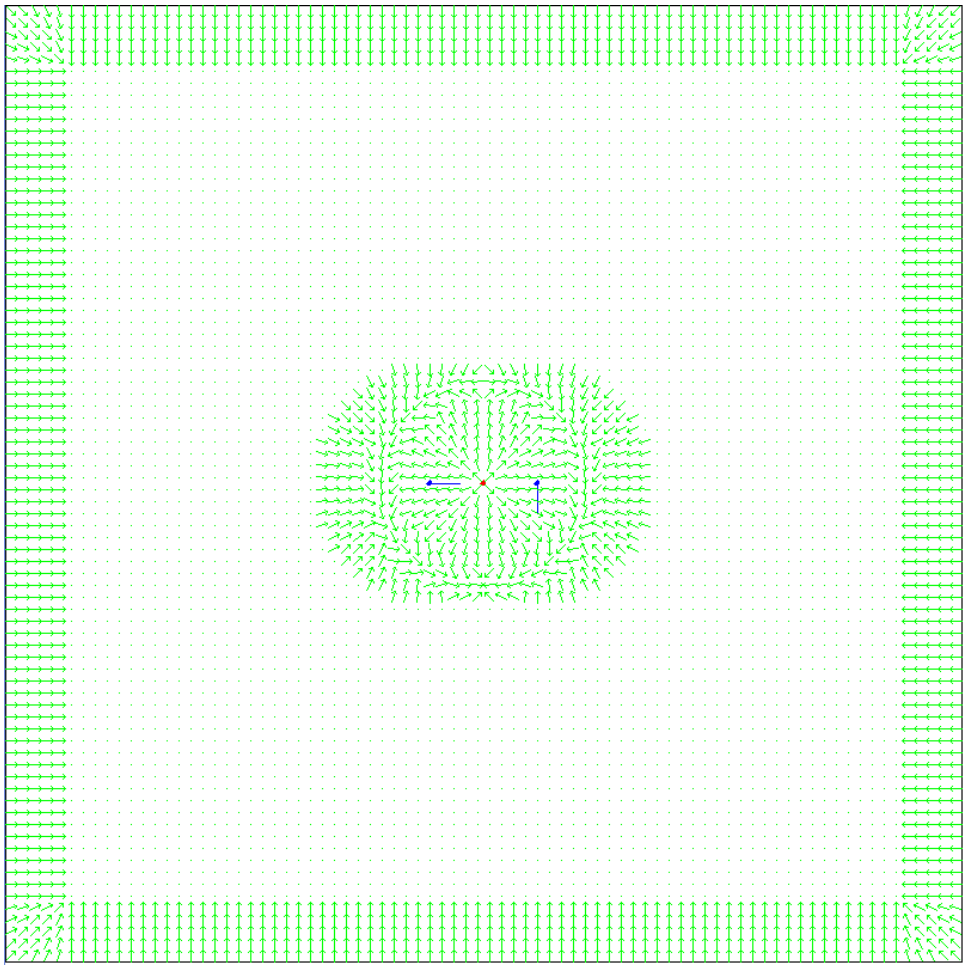

Figure C.2: Field plot for combining orbiting, flat target attraction, and flat target repulsion. UAVs at (380, 400) and (420, 400). There is a target at (400, 400). Plot assumes a sensor radius of 10km and a velocity of (0,1). Plot also assumes that the UAV for which the plot is drawn is traveling (0,1).

$$U_{R_3} = \frac{\sum_{i=0}^{|N|}(N_i.P - U.P)(dist(U.P, N_i.P) - U.BA.r_1 * U.Sr) \begin{bmatrix} 0 & dist(U.P, N_i.P) \leq U.BA.r_1 * U.Sr \\ 1 & otherwise \end{bmatrix}}{|N|} \tag{C.7}$$

In Kadrovich's work, this rule and a separation rule were combined into a single rule. In this work, the individual rules were kept separate to enable alterations to the cohesive and separation rules independently. Rather than use the cohesion and separation equation designed by Kadrovich [40], these different aspects are separated

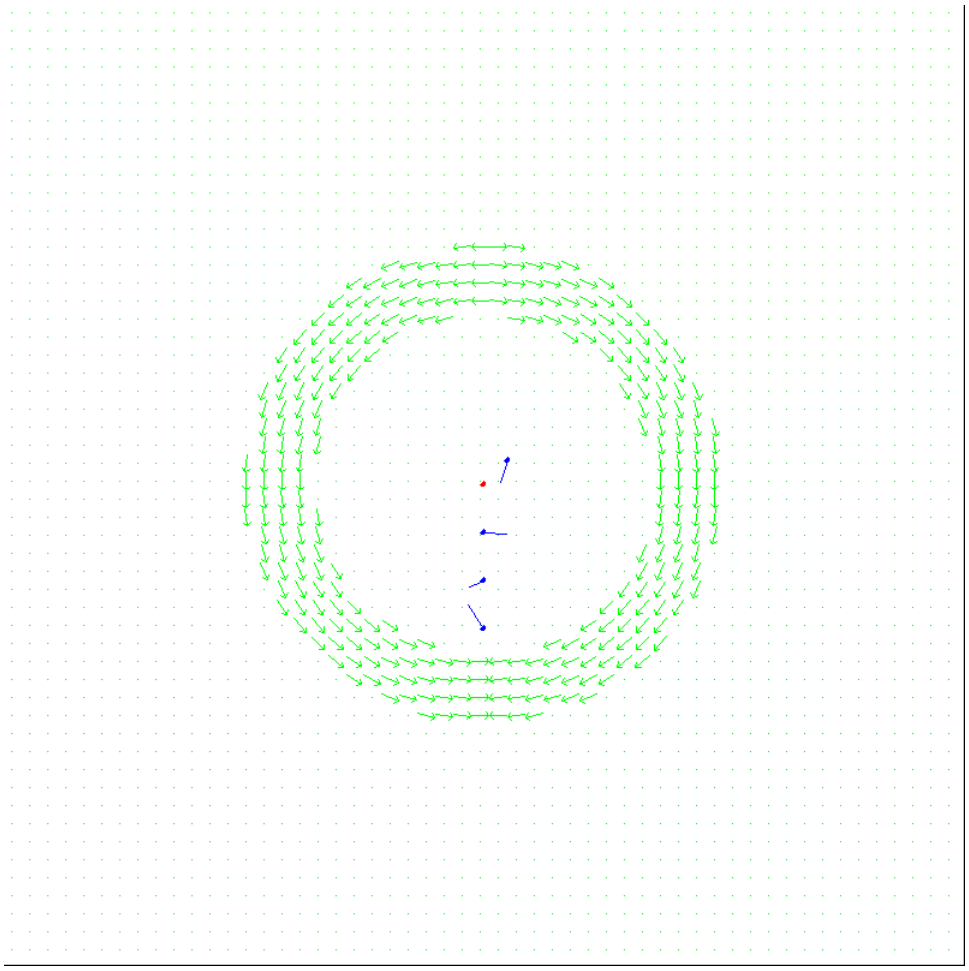

Figure C.3: Field plot for orbiting rule. UAVs at (420, 380), (400, 440), (400, 480), (400, 520) with individual velocities indicated by their direction line. Plot assumes a sensor radius of 30km and a velocity of (0,1).

to allow more flexible behavior evolution. The results of this rule can be seen in figure (C.4). In allowing separate weights for cohesion and separation independently, each particular rule can be independently addressed by the genetic algorithm. That is to say, the individual affects of cohesion or separation can be changed without necessarily changing the other.

As demonstrated in Figure (C.4), this behavior results in UAVs preferring to stay within a specified distance with other allied UAVs. This particular behavior rule has promise in preventing UAV formations from spreading out too far.

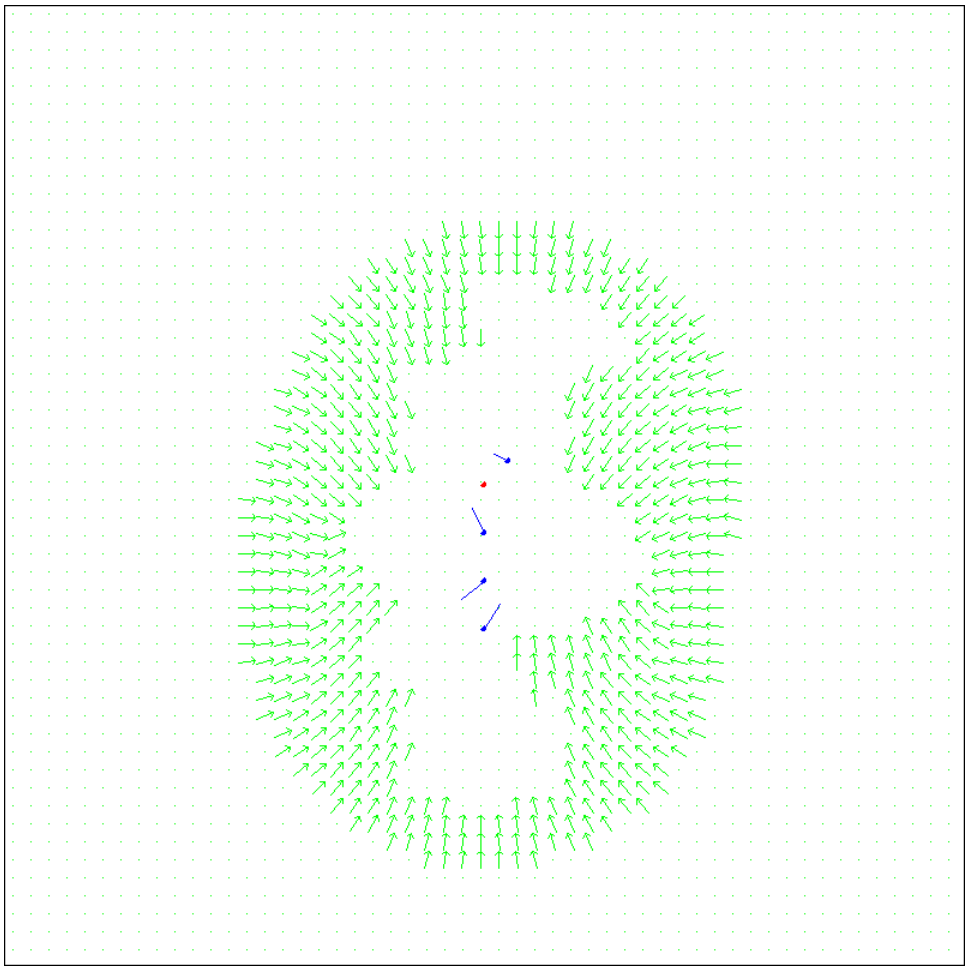

Figure C.4: Field plot for cohesion rule. UAVs at (420, 380), (400, 440), (400, 480), (400, 520) with individual velocities indicated by their direction line. Plot assumes a sensor radius of 30km and radius of 15km.

*C.1.4 Rule 4: Separation.* If UAV $U$ is too close to other UAVs, then there is a weight based repulsion similar to cohesion. The influence of repulsion is based upon how much closer other UAVs are to $U$ past a specified range, $U.Sr * U.BA.r_2$. This, too, was inspired by Kadrovich [40]. Equation (C.8) demonstrates how the separation rule is computed.

$$U_{R_4} = \frac{\sum_{i=0}^{|N|}(U.P - N_i.P)(U.BA.r_2 * U.Sr - dist(U.P, N_i.P)) \begin{bmatrix} 1 & dist(U.P, N_i.P) < U.BA.r_2 * U.Sr \\ 0 & otherwise \end{bmatrix}}{|N|} \tag{C.8}$$

The results of this rule can be seen in figure (C.5). Like the behavior for the cohesion rule, separation has a threshold of operation. Unlike cohesion, separation causes the UAVs to maintain a minimal distance to other UAVs. This means that separation has promise as a rule that can expand the sizes of UAV formations. Figure (C.5) demonstrates the effects of this rule.

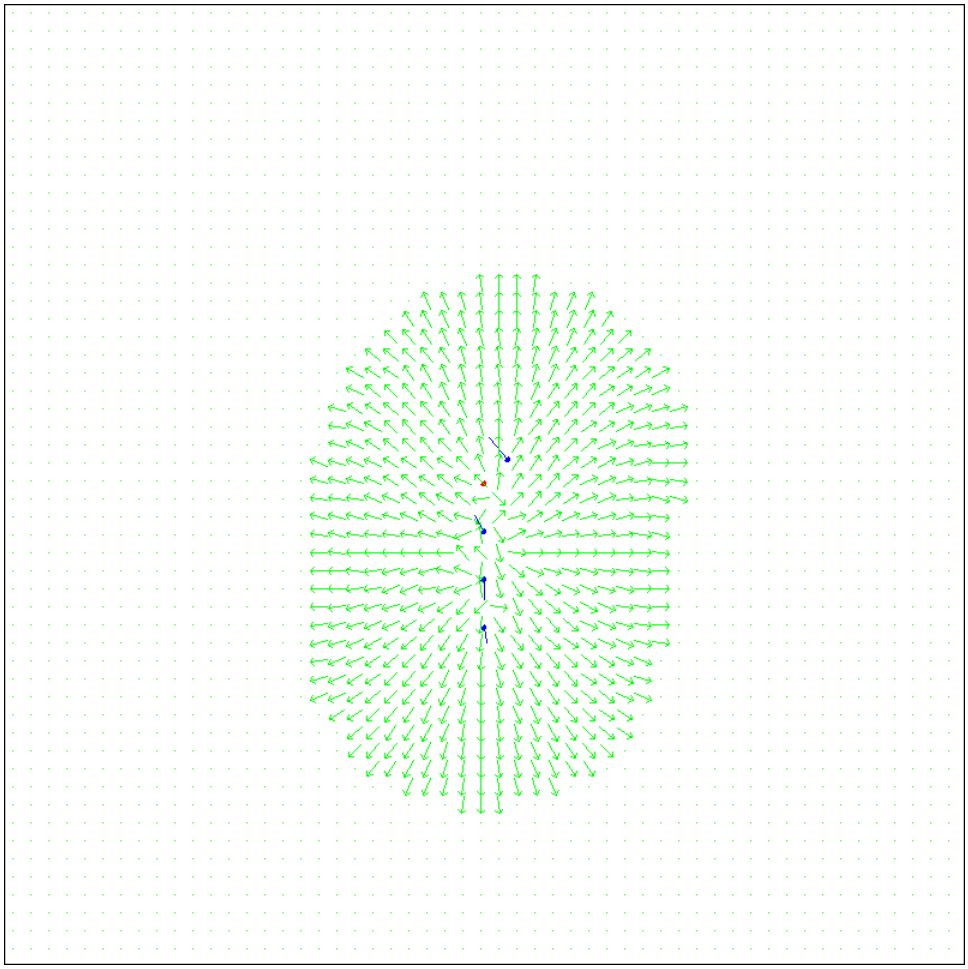

Figure C.5: Field plot for separation rule. UAVs at (420, 380), (400, 440), (400, 480), (400, 520) with individual velocities indicated by their direction line. Plot assumes a sensor radius of 30km and radius of 15km.

*C.1.5 Rule 5: Weighted Target Attraction.* UAVs are attracted to targets based upon the distance to said target. That is, UAVs proceed towards closer targets rather than further away targets.

$$U_{R_5} = \begin{bmatrix} \dfrac{\sum_{i=0}^{|\hat{T}|} \dfrac{T_i.P - U.P}{dist(U.P, T_i.P)^5}}{|\hat{T}|} & |\hat{T}| > 0 \\ \dfrac{\sum_{i=0}^{|N|} \dfrac{N_i.p(N_i.P - U.P)}{dist(U.P, N_i.P)}}{|N|} & otherwise \end{bmatrix} \tag{C.9}$$

Experimentation in [59] demonstrated the need for a weighted version of target attraction. The purpose for the weighted component is to cause the UAVs to proceed towards specific targets rather than towards the center of a target formation. Unweighted target attraction behaviors cause UAVs to move towards the target center of mass. This behavior is not detrimental when a UAV encounters a single target - the center of mass is that target. However, when multiple targets are known to exist, the target center of mass is between the targets and in a place at which the UAV may not be able to actually attack. For this reason, the behavior rule used for target attacking must provide some way to break the multi-target detection deadlock. The approach taken here is that the UAV attacks the closer target. Other weighting schemes may be of more use with other simulations. However, since the targets are homogeneous, they are all equal with respect to system performance. The preference towards attacking closer targets with this rule can be seen in Figure (C.6)

*C.1.6 Rule 6: Flat Target Repulsion.* UAVs are repelled from targets if they are within a 90% of their sensor range. The repulsion effect is uniform across all visible targets. The range prior to activation is geared to allow this rule to operation in conjunction to the target orbiting rule. Flat target repulsion is calculated in Equations (C.10).

$$U_{R_6} = \frac{\sum_{i=0}^{|\hat{T}|}(U.P - T_i.P) \begin{bmatrix} 1 & dist(U.P, T_i.P) < .9U.Sr \vee dist(U.P, T_i.P) < T_i.Ar \\ 0 & otherwise \end{bmatrix}}{|\hat{T}|} \tag{C.10}$$

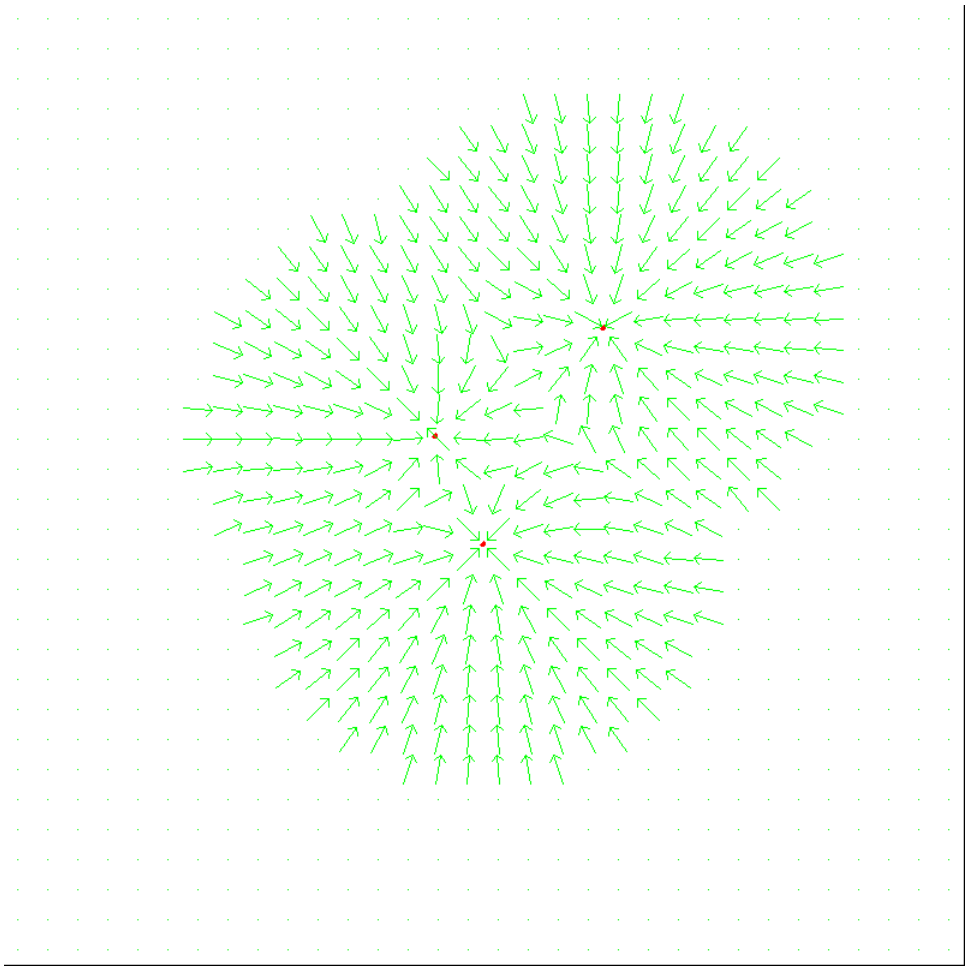

Figure C.6: Field plot for weighted target attraction rule. Targets at (360, 360), (400, 450), (500, 270). Plot assumes a sensor radius of 30km.

The purpose of the 90% range before execution is to allow UAVs to observe targets without necessarily being repulsed by them. This specific range effect is intended to allow this rule to operate in conjunction with the orbiting and flat target attraction rules as seen in Figure (C.2). A graphical representation of this rule operation can be seen in figure (C.7).

*C.1.7 Rule 7: Weighted Target Repulsion.* Each UAV is repelled from targets if they are within a particular range. The amount of repulsion for each UAV is based upon how close each UAV is to each target. UAVs are more repelled from

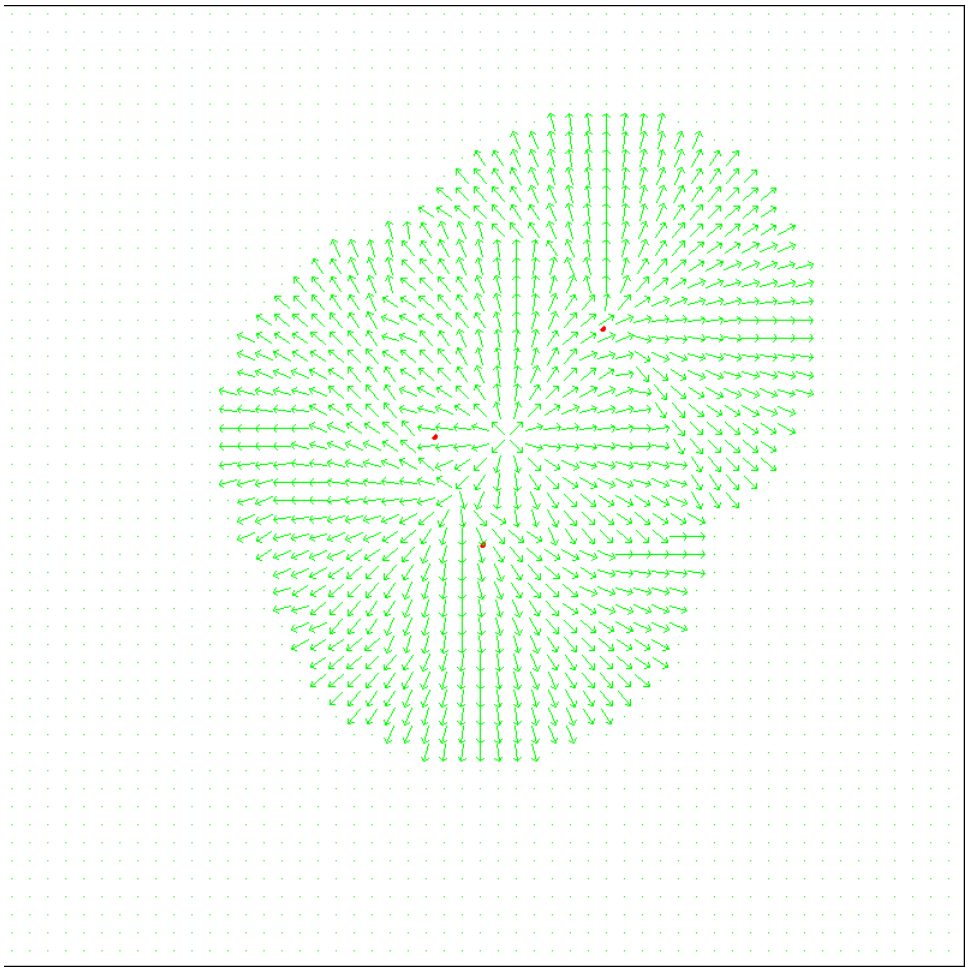

Figure C.7: Field plot for target repulsion rule. Targets at (360, 360), (400, 450), (500, 270). Plot assumes a sensor radius of 30km.

close targets than they are targets that are far away. Equation (C.11) demonstrates how this behavior rule is calculated.

$$U_{R_7} = \frac{\sum_{i=0}^{|\hat{T}|} \begin{bmatrix} \frac{(U.P - T_i.P)}{(U.BA.r_3 * U.Sr - dist(U.P, T_i.P)) \cdot 2} & |\hat{T}| > 0 \wedge dist(U.P, T_i.P) < U.BA.r_3 * U.Sr \wedge \\ & U.BA.r_3 * U.Sr > T_i.Ar \\ \frac{(U.P - T_i.P)}{(T_i.Ar - dist(U.P, T_i.P)) \cdot 2} & |\hat{T}| > 0 \wedge dist(U.P, T_i.P) < T_i.Ar \\ 0 & otherwise \end{bmatrix}}{|\hat{t}|} \tag{C.11}$$

This particular rule is distance weighted to cause the UAVs to be more repulsed by individual targets rather than the center of a target formation. The difference here is that repulsion from the target center of mass may cause a UAV to enter into a different target's engagement range rather than safely avoid the targets. A graphical representation of this rule operation can be seen in figure (C.8).

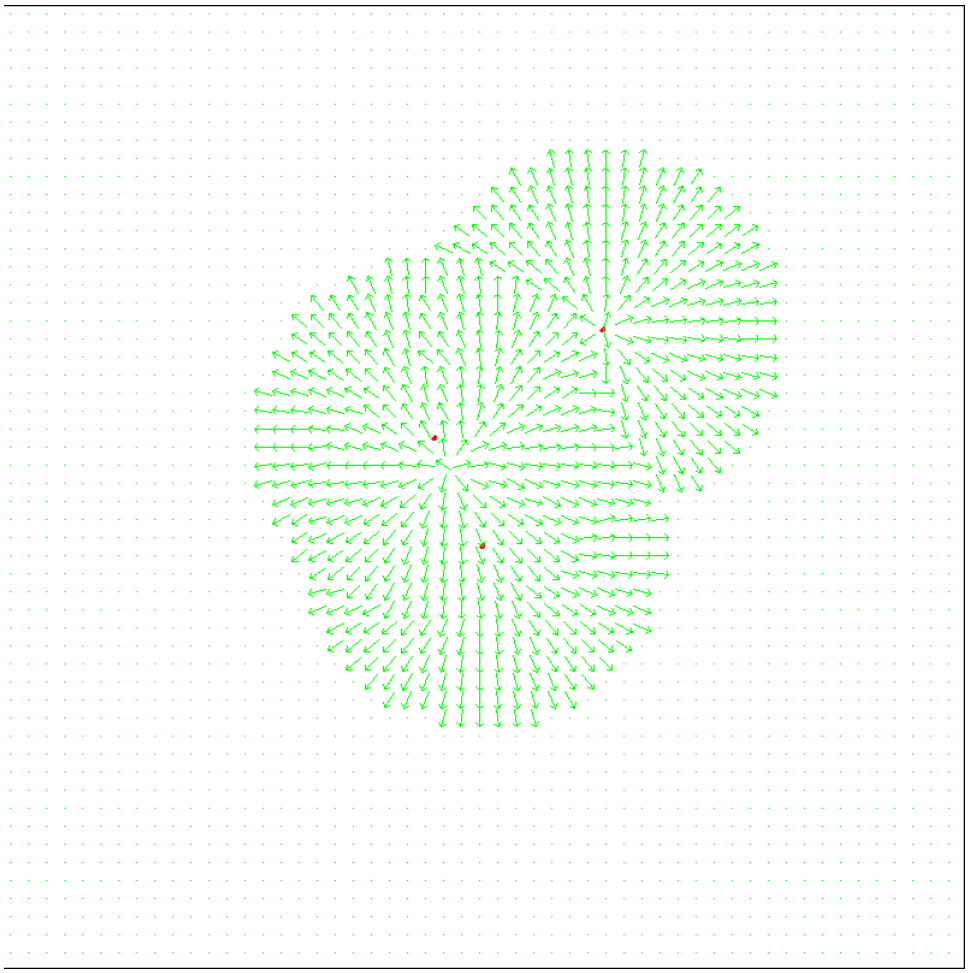

Figure C.8: Field plot for weighted target repulsion rule. Targets at (360, 360), (400, 450), (500, 270). Plot assumes a sensor radius of 30km and threshold radius of 15km.

*C.1.8 Rule 8: Flat Attraction.* UAVs proceed towards the center of mass for all known targets while they are outside of a given range with the target. This center mass is not necessarily close to any particular target. This rule, calculated

in Equation (C.12) is intended to keep the UAVs within a distance to the targets without creating a situation of undo risk.

$$U_{R_8} = \begin{bmatrix} \sum_{i=0}^{|\hat{T}|} T_i.P - U.P & |\hat{T}| > 0 \wedge dist(U.P, T_i.P) \geq .8U.Sr \\ \sum_{i=0}^{|N|} (N_i.P - U.P) & otherwise \end{bmatrix} \tag{C.12}$$

This rule is intended to cause UAVs to stop searching when they locate a target and stay within a 80% sensor range distance to a target to facilitate coordinated attacks. Like the constant weighting provided to the orbiting and flat target repulsion rules, the 80% range is intended to create a maximal range of minimum range of operation. Additionally, the constant weighting, set as it is, can combined with target orbiting and flat repulsion to create very stable safe orbits around a target as seen in Figure (C.2). A graphical representation of the flat attraction rule can be seen in figure (C.9).

*C.1.9 Rule 9: Evasion.* UAVs move away from each other if their next positions are too close. In this case, too close is determined to be 3 times the size of UAVs. This rule is inspired by Crowther [24]. However, unlike his definition, this particular implementation has application in all directions rather than simply in front of the UAV. This rule greatly increases the survivability of UAVs during simulation by causing them to avoid situations in which UAVs come too close.

The distance between the UAVs is calculated and truncated to a minimum value of one in Equation (C.13). This supports multiplicative weights later in Equation (C.15).

$$nDist(U.P, P) = \begin{bmatrix} dist(U.P, P) & dist(U.P, P) > 1 \\ 1 & otherwise \end{bmatrix} \tag{C.13}$$

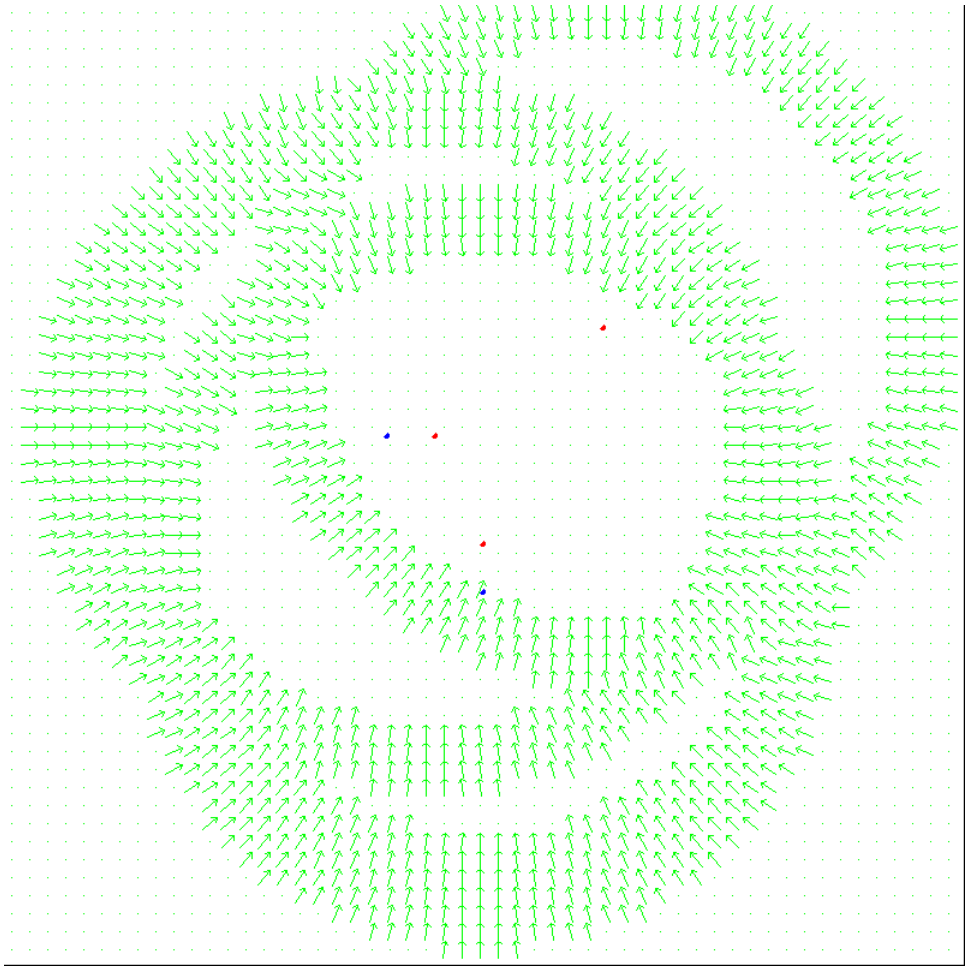

Figure C.9: Field plot for weighted target repulsion rule. Targets at (360, 360), (400, 450), (500, 270) and UAVs at (320. 360) and (400, 490). Plot assumes a sensor radius of 30km.

Next, projected future distance is computed based upon current direction and position. This important calculation, performed in Equation (C.14), is used to determine if the evasion rule is activated in Equation (C.15).

$$fDist(U,T) = dist(U.P + U.D, T.P + T.D) \tag{C.14}$$

Finally, the combined close proximity repulsion are summed for each known UAV. In summing the individual evasion values for each UAV, a vector describing the safest direction to evade towards is generation in Equation (C.15).

$$U_{R_9} = \frac{\sum_{i=0}^{|N|} \begin{bmatrix} \frac{nDist(U.P,N_i.P)}{3*Size}(U.P - N_i.P) & fDist(U,N_i) < 3*Size \wedge \\ & fDist(U,N_i) < nDist(U.P,N_i.P) \quad | > \\ 0 & otherwise \end{bmatrix}}{|N|} \tag{C.15}$$

The design decision to implement 360 degree applicability rather than simply within a frontal angle like Crowther's implementation was due to a couple of reasons. First of which, checking within specific angles requires more computation. Secondly, the intended visual system for the UAVs already examines 360 degrees and is therefore not limited to a range within visual capabilities. Lastly, by allowing a large range of applicable directions, both involved UAVs can take action to avoid a catastrophic impact. By only applying evasion to the frontal visual range like in [24], only the UAVs which detect possible impacts in the frontal range take action.

Additionally, the activation of this rule upon future state positions prevents too close positions in the future rather than present. If the rule were triggered by current proximities, then it may already be too late to prevent a collision!

*C.1.10 Rule 10: Obstacle Avoidance.* UAVs are repelled from obstacles based on two factors: whether the UAV's direction intersects the obstacle and proximity to the obstacle. Obstacle Avoidance causes the UAV to move in a direction parallel to the obstacle if the UAV's course intersects it. The weight of this direction parallelization is based upon how sharply the UAV intersectst the obstacle. If the angle is sharp, then parallization is minimal. Contrary to the parallization, each UAV is repulsed from an obstacle if it is closer than half its sensor range.

The distance between a UAV and an object are computed based upon the closest point between that UAV and the object. This is either an end point or the intersection of a perpendicular line from the UAV to the object. The distance weighting between

the UAV and its proximity is computed by comparison to the sensor range and the distance to the closest point on the target. This is accomplished in Equation (C.16).

$$d(U,O) = U.Sr - dist(U.P, O^U.Cp) \tag{C.16}$$

Additionally, the sum of all distances between the UAV and known obstacles is calculated in Equation (C.17). This is done to aid in a distance based waiting for the total behavior in Equation (C.21).

$$distSum = \sum_{i=0}^{|O|} d(U,O_i) \tag{C.17}$$

$$U_{R_{10}part1} = \begin{bmatrix} OVect(O_i,U)\frac{\angle(U.D-U.P)+\angle(O_i.Cp-U.P)}{90} & \angle(U.D-U.P)+\angle(O_i.Cp-U.P) < 90 \\ & \wedge O_i.Cp = inter(U,O_i) \wedge O_i.Line \\ 0 & otherwise \end{bmatrix} \tag{C.18}$$

$$OVect(O,U) = \begin{bmatrix} O.P_1 - O.P_2 & \angle(U.D-U.P)+\angle(O.P_1-O.P_2) < \\ & \angle(U.D-U.P)+\langle(O.P_2-O.P_1) \\ O.P_2 - O.P_1 & otherwise \end{bmatrix} \tag{C.19}$$

$$U_{R_{10}part2} = \begin{bmatrix} -\frac{U.Sr-dist(U.P,O_i.Cp)}{U.Sr}(O_i.Cp-U.P) & dist(U.P,O_i.Cp) < U.Sr/2 \\ 0 & otherwise \end{bmatrix} \tag{C.20}$$

$$U_{R_{10}} = \sum_{i=0}^{|\hat{O}|} \frac{U_{R10,part1}+U_{R10,part2}}{\frac{distSum}{d(U,O_i)}} \tag{C.21}$$

A graphical depiction of this rule's effect is in figure (C.10). For the most part, this rule keeps UAVs safe by providing a repulsion. As UAVs get closer to an obstacle, this rule provides a way in which the UAVs avoid hitting the object.

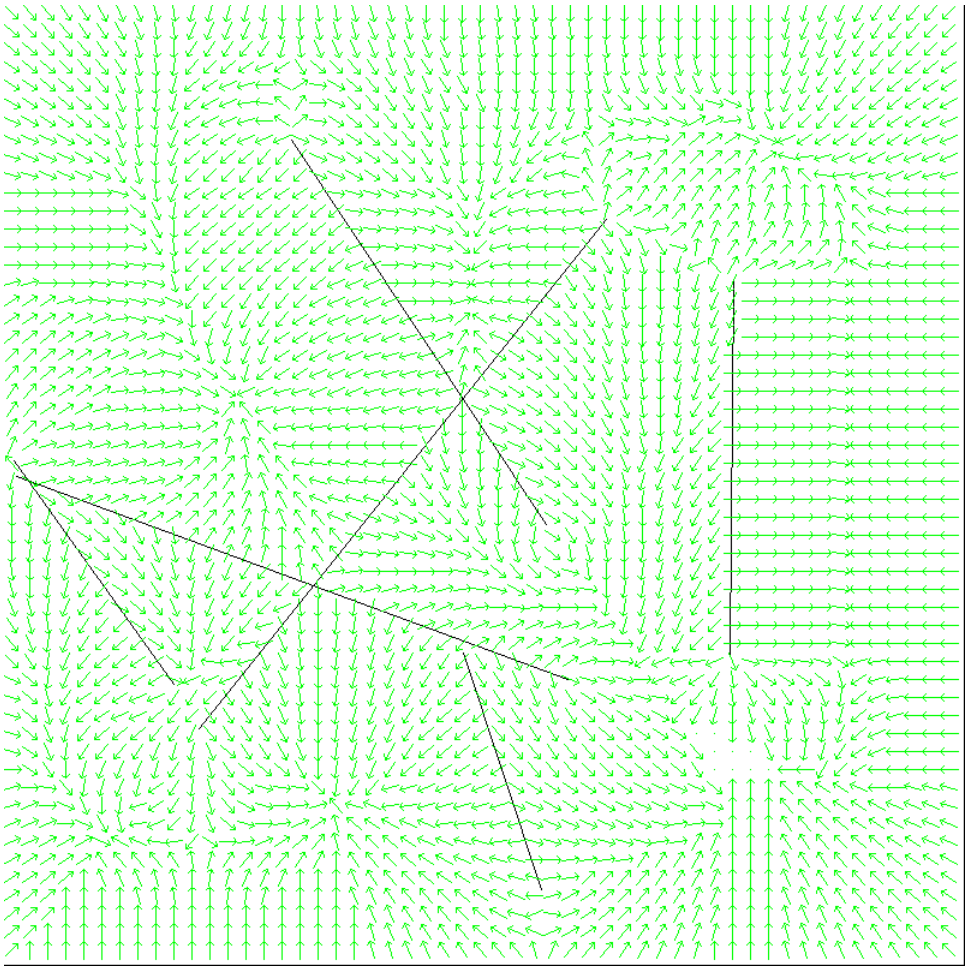

Figure C.10: Field plot for obstacle avoidance. Obstacles are randomly generated. Plot assumes a sensor radius of 30km and velocity of (0,1).

*C.1.11 Rule Summation and Normalization.* The way in which the rules are combined is significant. This is because it changes the influence each behavioral rule bears upon the final direction a UAV takes. In this investigation, the rules are weighted by the behavioral archetype values and summed. With respect to the safety rules, evasion and obstacle avoidance, their weights are hard-coded at twice the maximal weight for normal rules. Equation (C.22) demonstrates how the rules are combined.

$$U.D_{new} = (\sum_{i=1}^{8} U_{R_i} \frac{U.BA.W_i}{U_{R_i}.length}) + (\sum_{i=9}^{10} U_{R_i} \frac{2}{U_{R_i}.length}) \tag{C.22}$$

Equation (C.22) demonstrates how the various behavior rules are combined. This is accomplished through a weighted summation. Within the first summation, the first 8 behavior rules are combined. These rules are allowed to evolve within the system. Additionally, the values derived from each rule are normalized to a unit vector. This is performed so that the results of all rules, when combined with their behavior archetype weight fall within a [0.0...1.0] interval. The second summation functions similarly to the first. It addresses behavior rules 9 and 10 which are important for UAV safety. These rules are normalized to a vector of length 2. This is done to allot more behavioral influence, regardless of evolutionary attributes, to the safety rules.

Other potential ways to combined the rules include just adding their weighted components without normalizing the rule based upon its length. When the rules are summed without prior normalization, rules with longer vector results have undo influence upon the UAVs next behavior. That is to say, if a particular rule returns a direction vector that is much larger than the others, then it has potentially unwarranted influence upon the system. Without early normalization, the rules with longer resulting vectors tend to overwhelm the more subtle rules.

```
package swf.uav.Behaviors;

import java.util.ArrayList;

import swf.core.MathVector;
import swf.core.StaticValues;
//import swf.fileMan.logFileTester;
import swf.uav.NeighborhoodAgent;
import swf.uav.NeighborhoodTarget;
import swf.uav.UAV;

/**
 * This method uses the three step process that bees use to choose a new hive.
 * First recon, second recruit, third threshold choose. Inspiration from
 * Visscher's paper entitled "Choosing a home: how the scouts in a honey bee
 * swarm perceive the completion of their group decision making"
 *
 * @author dnowak
 *
 */
public class BeeAttack extends BehaviorTemplate {

    boolean votedFlag = false;

//  private logFileTester log = new logFileTester("tracker.txt");

    UAV closestTarget = UAVLink;
```

```
/**
 * the contstruct for the attack sequence
 *
 * @param uav
 */
public BeeAttack(UAV uav) {
    super(uav);
    numberInputWeights = 2;
}

/*
 * (non-Javadoc) If the poisition in the attack holder from UAV is UAVs
 * position then the agent is in a state without a current target If the
 * choosenTarget is empty the behavior does not have enough information to
 * make a deicision It needs to do more Recce otherwise Move torawrds
 * intended target
 *
 * @see swf.uav.Behaviors.BehaviorTemplate#calculateVector()
 */
@SuppressWarnings("static-access")
@Override
protected MathVector calculateVector() {


    initializeBeeAttack();
    analyzeTargetArea();
    if (choosenTarget == null) {
        returnVector = recceTargetArea();
    } else {
```

```
        returnVector = moveTorwardsTarget();
    }
    return returnVector;
}

@SuppressWarnings("static-access")
private void initializeBeeAttack() {
    if (!(choosenTarget == null) && !choosenTarget.isAlive()) {
        choosenTarget = null;
    }
    if (choosenTarget == null) {
        votedFlag = false;
    }
    if (choosenTarget != null){
    }

    // set the subcontroller wieght to the ratio of the sub swarm size
    subControllerWeight = ((subControllerWeight + 17) / 34)
            * UAVLink.getNeighbors().size();

    // search for the closest target
    double bestDist = Double.MAX_VALUE;
    for (NeighborhoodTarget workTarget : UAVLink.getNeighborTargets()) {
        UAV tempTarget = workTarget.getTarget();

        double testDist = tempTarget.getDistanceTo(UAVLink);
        if (testDist < bestDist && tempTarget.isAlive()) {
            closestTarget = tempTarget;
            bestDist = testDist;
```

```
        }
    }

    if(UAVLink.equals(closestTarget) ||(UAVLink.getNeighborTargets().equals(nul
        UAVLink.setRecceFlag(false);
        UAVLink.setReconKeyPost(new MathVector[2]);
    }

}

/**
 * Look at all of the targets in the list, find the closest and continue a
 * recce loop around it -- in theory it should bounce from target to target
 * if the engagement rings intersect
 *
 * @return MathVector - un-normalize heading vector
 */
@SuppressWarnings("static-access")
private MathVector recceTargetArea() {
    MathVector recceVector = new MathVector(StaticValues
            .getDomainDimension());



    // save the position and direction and move into recce state
    if (!closestTarget.equals(UAVLink) && !UAVLink.isRecceFlag()) {
        MathVector[] reconKeyPost = new MathVector[2];
        reconKeyPost[0] = new MathVector();
        reconKeyPost[0] = UAVLink.getPosition();
```

```
            reconKeyPost[1] = new MathVector();
            reconKeyPost[1] = UAVLink.getDirection();
            UAVLink.setReconKeyPost(reconKeyPost);
            UAVLink.setRecceFlag(true);

        }

        // move towards target if it slips outside of sensor range
        MathVector recceMoveCheck = (closestTarget.getPosition().sub(UAVLink.getPos
        if (recceMoveCheck.getLength() > UAVLink.getSensorRange()){
            recceVector = recceMoveCheck;
        } else {
        // sets the orbit as the closest target or none if it is itself
        recceVector = WeightTargetOrbit.helperOrbit(UAVLink, closestTarget);
        }


        if(!UAVLink.isTarget()&& !UAVLink.equals(closestTarget)){
            @SuppressWarnings("unused")
            double dirctTo = closestTarget.getPosition().sub(UAVLink.getPosition()
        }

        return recceVector;
    }

    /**
     *
     * looks to see if the UAV has reached its initial target detection point It
     * is known at the original spot after moving around the target area if
```

```
 * three things are true: Original position in the sensor The direction is
 * within 45* of the original direction and Original position in front
 */
@SuppressWarnings("static-access")
private void analyzeTargetArea() {
    // add voted flag
    if (UAVLink.isRecceFlag()) {
        double diffDir = ((Double) UAVLink.getDirection().sub(
                UAVLink.getReconKeyPost()[1]).getLength()).doubleValue();
        diffDir = Math.abs(diffDir);
        double diffPos = ((Double) UAVLink.getPosition().sub(
                UAVLink.getReconKeyPost()[0]).getLength()).doubleValue();
        diffPos = Math.abs(diffPos);
        double directionTo = (UAVLink.getReconKeyPost()[0].sub(UAVLink
                .getPosition())).findAngle()
                - UAVLink.getDirection().findAngle();
        directionTo = Math.abs(directionTo);
        if (diffPos < UAVLink.getSensorRange() && diffDir < 45.0
                && (directionTo >= 300 | directionTo <= 60)) {
            UAVLink.setReconKeyPost(null);
            UAVLink.setRecceFlag(false);
            UAV workingTarget = analyzeTargetSet();
            choosenTarget = workingTarget;
        }
    }
}

/**
 * looks at the current information of the target area and determines the
```

```
 * most oppurtune target
 *
 * @return UAV
 */
private UAV analyzeTargetSet() {
    UAV votedTarget = null;
    ArrayList<UAV> tempTargetList = new ArrayList<UAV>();
    double bestDist = Double.MAX_VALUE;
    double leastOverlap = Integer.MAX_VALUE;

    // count the number of overlapping engagement rings
    for (NeighborhoodTarget workTarget : UAVLink.getNeighborTargets()) {
        UAV tempTarget = workTarget.getTarget();
        int tempOverlap = getOverlappingEngagement(tempTarget);
        if (tempOverlap < leastOverlap) {
            tempTargetList.clear();
            tempTargetList.add(tempTarget);
        } else if (tempOverlap == leastOverlap) {
            tempTargetList.add(tempTarget);
        }
    }

    // choose the closets of the weakest targets
    if (tempTargetList.size() > 0) {
        for (UAV tempTarget2 : tempTargetList) {
            double workDist = ((Double) tempTarget2.getPosition().sub(
                    UAVLink.getPosition()).getLength()).doubleValue();
            if (bestDist > workDist) {
                votedTarget = tempTarget2;
```

```
            }
        }
    }
    return votedTarget;
}

/**
 * @param tempTarget
 * @return in - number of overlapping engagement rings on that target
 */
private int getOverlappingEngagement(UAV tempTarget) {
    int returnNum = 0;
    for (NeighborhoodTarget workTarget : UAVLink.getNeighborTargets()) {
        UAV tempTargetNeighbor = workTarget.getTarget();
        double distTo = ((Double) tempTargetNeighbor.getPosition().sub(
                tempTarget.getPosition()).getLength()).doubleValue();
        if (distTo < tempTargetNeighbor.getSensorRange()) {
            returnNum++;
        }
    }
    return returnNum;
}

/**
 * If given the clearance to move towards a target Goes towards if the
 * attack numbers are approipriate or loiters at perimeter waiting for other
 * to join attack
 *
 * @return MathVector
```

```
 */
@SuppressWarnings("static-access")
private MathVector moveTorwardsTarget() {

    MathVector direction = choosenTarget.getPosition().sub(
            UAVLink.getPosition());
    double dist = direction.getLength();
    int numberVoted = getVotingNeighbor(choosenTarget);
    if ((UAVLink.getAttackRange() > dist)
            || (numberVoted > subControllerWeight)) {
        returnVector = direction;
    } else {
        returnVector = WeightTargetOrbit
                .helperOrbit(UAVLink, choosenTarget);
    }
    return returnVector;

}

/**
 * Looks at the neighborhood and determines who else is attacking the same
 * target and returns the count.
 *
 * @return int - number of agents attack same target
 */
private int getVotingNeighbor(UAV choosenTarget) {
    int returnCtr = 0;
    for (NeighborhoodAgent workingAgent : UAVLink.getNeighbors()) {
        if (workingAgent.getAgent().getVoted() == choosenTarget) {
```

```
                returnCtr++;
            }

        }
        return returnCtr;
    }

}
```

```
package swf.uav.control;

import java.util.BitSet;
import java.util.Vector;

import swf.core.DataBlackboard;
import swf.core.MathVector;
import swf.core.StaticValues;

/**
 * This class attempts to replace the NN control that orginially existed in the
 * simulation. As a result there is artifact code.
 *
 * The actual control attempts to determine based on sensory information the
 * abstract state. The process is similar to DE but used in the input spaces as
 * opposed to the solution spaces. As such the position and second order
 * variables of the state are controlled and optimized by the GA. (similar to
 * the NN implementation)
 *
 * All inputs of sensory information must be normalized.
 *
 * @author dnowak
 *
 */
public class BehaviorMatrixDE extends BehaviorMatrix {

    private double randomFactor = .5;
```

```
int returnState = 0;

/**
 * @param in
 * @param genes
 */
public BehaviorMatrixDE(double[] in, BitSet genes) {
    super(in, genes);

}

/**
 * Arifact code (non-Javadoc)
 *
 * @see swf.uav.control.BehaviorMatrix#nextStateD(double, double, double[])
 */
@Override
public int nextStateD(double s1, double s2, double[] s3) {

    return returnState;
}

/**
 * Arifact code (non-Javadoc)
 *
 * @see swf.uav.control.BehaviorMatrix#nextStateP(double, double, double[])
 */
@Override
public int nextStateP(double s1, double s2, double[] s3) {
```

```
    int tempNextD = nextStateD(s1, s2, s3);

    if (Math.abs(DataBlackboard.rand.nextDouble()) > randomFactor) {
        tempNextD = DataBlackboard.rand.nextInt(StaticValues
                .getNumbBuckets());
    }
    returnState = tempNextD;

    return returnState;
}



/**
 * this implemenation looks at only the weight euclidean distance
 * which is more precise than the coverage approach
 * (non-Javadoc)
 *
 * @see swf.uav.control.BehaviorMatrix#nextStateP(double[])
 */
@Override
public int nextStateP(double[] inputSensors) {
    int tempNextD = nextStateD(inputSensors);

    // randomize
    if (Math.abs(DataBlackboard.rand.nextDouble()) > randomFactor) {
        tempNextD = DataBlackboard.rand.nextInt(StaticValues
                .getNumbBuckets());
```

```
    }
    returnState = tempNextD;

    return returnState;
}

/**
 * This implemenation looks at only the weight euclidean distance
 * which is more precise than the coverage approach
 * (non-Javadoc)
 *
 * @see swf.uav.control.BehaviorMatrix#nextStateD(double[])
 */
@Override
public int nextStateD(double[] inputSensors) {
    return getBestBA( inputSensors);

}

/**
 * Chooses that BA from all the behaviors
 * @param inputSensors
 * @return int
 */
private int getBestBA(double[] inputSensors) {
    int numbSens = StaticValues.getNumbSenses();
    int numBucks = StaticValues.getNumbBuckets();
    // build MathVector for sensor inputs
    MathVector inputSensorMV = new MathVector(inputSensors);
```

```
double bestBA = Double.MAX_VALUE;
int keeper = 0;

// loop through all of the buckets
for (int testBA =0; testBA < numBucks; testBA++) {
    double[] tempbaVal = new double[numbSens / 2];
    double[] tempbaWeights = new double[numbSens / 2];
    int placeHold = testBA * numBucks;
    int ctr = 0;
    // build array of doubles for that BA control weights
    for (int workNum = placeHold; workNum < placeHold + (numbSens); workNu
        tempbaVal[ctr] = componentWeights[workNum];
        tempbaWeights[ctr] = componentWeights[workNum+1];
        ctr++;
    }
    // build math vector from that list
    MathVector bAvalues = new MathVector(tempbaVal);
    MathVector weightVec = new MathVector(tempbaWeights);

    // get distance
    double testReading = inputSensorMV.weightEuclideanDist(bAvalues, weigh

    // test distance best
    if (testReading < bestBA) {
        bestBA = testReading;
        keeper = testBA;
    }
}
```

```
    return keeper;
}

/**
 * takes all the normalize sensor inputs and expands them out to the size of
 * the optimization controls in the BA as governed by the number of bits in
 * the genotype
 *
 * @param inputSensors
 * @return double[]
 */
@SuppressWarnings("unused")
private double[] scaleSensorInputs(double[] inputSensors) {

    int inputDomainSize = ((Double) Math.pow(2, StaticValues
            .getNumbBuckets())).intValue();

    for (double workingInput : inputSensors) {
        workingInput = (workingInput * inputDomainSize)
                - (inputDomainSize / 2);
    }
    return inputSensors;
}

/**
 * returns the winning behavior by chosing from the set of behaviors
 * in the winner vector
 * @param winner
 * @param inputSensors
```

```
 * @return int
 */
@SuppressWarnings("unused")
private int getBestBA(Vector<Integer> winner, double[] inputSensors) {
    int numbSens = StaticValues.getNumbSenses();
    int numBucks = StaticValues.getNumbBuckets();
    // build MathVector for sensor inputs
    MathVector inputSensorMV = new MathVector(inputSensors);
    double bestBA = Double.MAX_VALUE;
    int keeper = 0;
    // test each BA against the inputs
    for (int testBA : winner) {

        double[] tempbaVal = new double[numbSens / 2];
        int placeHold = testBA * numBucks;
        int ctr = 0;
        // build array of doubles for that BA control weights
        for (int workNum = placeHold; workNum < placeHold + (numbSens / 2); wo
            tempbaVal[ctr] = componentWeights[workNum];
            ctr++;
        }
        // build math vector from that list
        MathVector bAvalues = new MathVector(tempbaVal);

        // get distance
        double testReading = inputSensorMV.euclideanDist(bAvalues);

        // test distance best
        if (testReading < bestBA) {
```

```
            bestBA = testReading;
            keeper = testBA;
        }
    }

    return keeper;
}

/**
 * Check to see if that BA cover the input state
 *
 * @param i
 * @param inputSensors
 * @return boolean
 */
@SuppressWarnings("unused")
private boolean checkBucket(int i, double[] inputSensors) {
    int tracker = 0;
    // int numSense = StaticValues.getNumbSenses();
    for (double workingValue : inputSensors) {
        int workingConVal = tracker;
        if (!(workingValue < componentWeights[workingConVal]
                + componentWeights[workingConVal + 1])
                || !(workingValue > componentWeights[workingConVal]
                        - componentWeights[workingConVal + 1])) {
            return false;
        }
        tracker += 2;
    }
```

```
            return true;
        }

    }
```

# *Bibliography*


1. Abbass, H. "Self-adaptive pareto differential evolution", 2002. URL citeseer.ist.psu.edu/abbass02selfadaptive.html.

2. Abbass, Hussein A., Ruhul Sarker, and Charles Newton. "PDE: A Pareto-frontier Differential Evolution Approach for Multi-objective Optimization Problems". *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, 971–978. IEEE Press, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. ISBN 0-7803-6658-1. URL citeseer.ist.psu.edu/abbass01pde.html.

3. Aerovironment. "Raven RQ-11B". Datasheet, June 2007. URL http://www.aerovironment.com/.

4. Agarwal, M.H. Lim & M.J. Er, A. "ACO for a New TSP in Region Coverage". *IEEE/RSJ International Conf. on Intelligent Robots & Systems (IROS),*. August 2005. Edmonton, Canada.

5. Alba, E. Ed. "The VRP Web". Network and Emerging Optimization, Mar 2007. URL http://neo.lcc.uma.es/radi-aeb/WebVRP/. Accessed 22 May 2007.

6. Australia, Team. "Simulation - UAV Simulator". Australian Department of Defense, 05 Oct 2006. URL http://www.defence.gov.au/teamaustralia/simulation_UAV_simulator.htm. Accessed 14 Feb 07.

7. Bäck, Thomas. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996. ISBN 0-19-509971-0.

8. Barlow, Gregory J. and Choong K. Oh. "Robustness analysis of genetic programming controllers for unmanned aerial vehicles". *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 135–142. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-186-4.

9. Barr, Bruce L. Golden James Kelly William R. Stewart Mauricio G.C. Resende, Richard S. "Guidelines for Designing and Reporting on Computational Experiments with Heuristic". *Proceedings of International Conference on Metaheuristics*. 2001.

10. Bernstein, Daniel S., Shiomo Zilberstein, and Neil Immerman. "The Complexity of Decentralized Control of Markov Decision Processes". *In Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, 32–37. 2000. URL citeseer.ist.psu.edu/article/bernstein00complexity.html.

11. Bonabeau, Eric et al. *Swarm Intelligence: From Natural to Artificial Systems.* Sante Fe Institute, 1999. New York.

12. Boutilier, Craig. "Sequential Optimality and Coordination in Multiagent Systems". *IJCAI*, 478–485. 1999. URL `citeseer.ist.psu.edu/article/boutilier99sequential.html`.

13. Braitenburg, Valentino. *Vehicles: Experiments in Synthetic Psychology*. Bradford Books, 1987.

14. Bush, George W. "Department of Defense 2007 Budget Report". White House release, 2007. URL `http://www.whitehouse.gov/omb/budget/fy2007/pdf/budget/defense.pdf`.

15. Camazine, Scott. *Self Organization in Biological Systems*. Princeton University Press, USA, 2003.

16. de Castro, Leandro Nunes and Fernando Jos Von Zuben. *Artificial Immune Systems: Part Ii - A Survey Of Applications*. Technical Report DCA-RT 02/00, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, SP, Brazil, February 2000. URL `citeseer.ist.psu.edu/nunesdecastro00artificial.html`.

17. CLARK, RICHARD M. *Uninhabited Combat Aerial Vehicles*. Air University Press, 401 Chennault Circle Maxwell Air Force Base, Alabama 36112-6615, 2000.

18. Clarke, Brig. Gen. Stanley and Col. Gail Wojtowicz. "The U.S. Air Force Remotely Piloted Aircraft and Unmanned Aerial Vehicle Strategic Vision". Air Force Link, 2005. URL `www.af.mil/shared/media/document/AFD-060322-009.pdf`.

19. Coello, Carlos A. Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. ISBN 0387332545.

20. Coello Coello, Carlos A. and Gary B. Lamont (editors). *Application of Multi Oobjective Evolutionary Algorithms*. World Scientific Publishing, 2004.

21. Cole, Brandon. "picture of schools of fish, Horse-eye Jacks, with scuba diver". Brandon Cole marine photography, Oct 2007.

22. Corner, Joshua. *Swarming Reconnaissance Using Unmanned Aerial Vehicles In A Parallel Discrete Event Simulation*. Master's thesis, Air Force Institute of Technology, 2004.

23. Cottam, R., W. Ranson, and R. Vounckx. "Autocreative hierarchy II: dynamics self-organization, emergence and level-changing". *Integration of Knowledge Intensive Multi-Agent Systems, 2003. International Conference on*, 766–773. 30 Sept.-4 Oct. 2003.

24. Crowther, W.J. "Flocking of autonomous unmanned air vehicles". *Aeronautical Journal*, 107(1068):99–110, February 2003.

25. Deb, K., Samir Agrawal, Amrit Pratap, and T. Meyarivan. "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II". Marc Schoenauer, K. Deb, G. Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel (editors), *Parallel Problem Solving from Nature – PPSN VI*, 849–858. Springer, Berlin, 2000.

26. Deb, K., L. Thiele, M. Laumanns, and E. Zitzler. "Scalable Test Problems for Evolutionary Multi-Objective Optimization". A. Abraham, R. Jain, and R. Goldberg (editors), *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, chapter 6, 105–145. Springer, 2005. ISBN 1-85233-787-7.

27. Dictionary, Meriam-Webster. "Automaton". URL `http://www.m-w.com/dictionary/automaton`. Def 1.

28. Dreo J., P. Siarry, A. Petrowski and E. Taillard. *Metaheuristics for Hard Optimization*. Springer, 2006.

29. Gat, E. "On three-layer architectures". R. P. Bonnasso D. Kortenkamp and editors R. Murphy (editors), *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, 1997.

30. Gmytrasiewicz, P. and P. Doshi. "Interactive pomdps: Properties and preliminary results", 2004. URL `citeseer.ist.psu.edu/gmytrasiewicz04interactive.html`.

31. Goktogan, E.; Ridley M.; Sukkarieh S., A.H.; Nettleton. "Real time Multi-UAV Simulator". *Proceedings of IEEE International Conference on Robotics and Automation*, 2:ICRA '03, 14-19 Sept. 2003. URL `agoktogan,ericn,m.ridley,salah@acfr.usyd.edu.au`.

32. Haken, Hermman. *Information and Self Organization*. Springer, Germany, 2003.

33. Hayes, Adam T. and Parsa Dormiani-Tabatabaei. "Self-Organized Flocking with Agent Failure: Off-Line Optimization and Demonstration with Real Robots". *In Proceedings of the 2002 IEEE International Conference on Robotics and Automation IROS-02*, 3900–3905. 2002.

34. Heylighen, Francis. "Self-organization and complexity in the natural sciences". Principia Cybernetica Web, Dec 2006. URL `http://pespmc1.vub.ac.be/COMPNATS.HTML`.

35. Heylighen, Francis and Cliff Joslyn. "Cybernetics and Second-Order Cybernetics". *Encyclopedia of Physical Science & Technology*, 3rd ed., 2001. Academic Press, New York.

36. Holland, John H. *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley, 1996. New York.

37. Hu, Xiaohui, Russell C. Eberhart, and Yuhui Shi. "Particle Swarm with Extended Memory for Multiobjective Optimization". *Swarm Intelligence Sympo-*

*sium, 2003. SIS '03. Proceedings of the 2003 IEEE*, 193 – 197. April 2003. URL citeseer.ist.psu.edu/hu03particle.html.

38. ICOSYSTEM. "Technology", 2002. URL http://www.icosystem.com/technology.htm. Accessed 14 Feb 2007.

39. Jumper, General John P. "Air Force Doctrine Document 1". United States Air Force, November 2003. URL http://www.dtic.mil/doctrine/jel/service pubs/afdd1.pdf.

40. Kadrovach, Tony. *Communications Modeling System For Swarm-Based Sensors.* Ph.D. thesis, Air Force Institute of Technology, 2003.

41. Khosla, D. and T. Nichols. "Hybrid evolutionary algorithms for network-centric command and control". *Defense Transformation and Network-Centric Systems. Edited by Suresh, Raja. Proceedings of the SPIE, Volume 6249, pp. 624902 (2006).*, volume 6249 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*. June 2006.

42. Kleeman, Mark. "Self Organization", 2005. Air Force Institute of Technology.

43. Lau, Henry Y. K., Vicky W. K. Wong, and Ivan S. K. Lee. "Immunity-based autonomous guided vehicles control". *Appl. Soft Comput.*, 7(1):41–57, 2007. ISSN 1568-4946.

44. Leon-Garcia, Alberto. *Probability and Random Processes for Electrical Engineering (2nd ed).* Addison Wesley, 1994.

45. Littman, Michael. "Markov Models For Sequential Decision Making". Computer Science Department Brown University, 1997. URL http://www.cs.duke.edu/ mlittman/topics/pomdp-page.html. University of Pennsylvania, Brandeis University, and the University at Stony Brook.

46. Lluch, Dan. "Multi UAV Simulation". MATLAB Central File Exchange, 22 Aug 2002. URL http://www.mathworks.com/matlabcentral/fileexchange/. Accessed 14 Feb. 2007.

47. Lua, Karl ALtenburg, Chin A. and Kendall E. Nygard. "Synchronized Multi-Point Attack by Autonomous Reactive Vehicles with Local Communication". *Proceedings of the 2003 IEEE Swarm Intelligence Symposium.* 2003.

48. METAVR. "MetaVR Provides Visual Systems for MUSE UAV Simulation Program", 2007. URL http://www.metavr.com/casestudies/museuav.html. Accessed 14 Feb. 2007.

49. Metsker, Steven John and William C. Wake. *Design Patterns in Java(TM) (2nd Edition) (Software Patterns Series).* Addison-Wesley Professional, 2006. ISBN 0321333020.

50. Milam, Kevin M. *Evolution of Control Programs for a Swarm of Autonomous Unmanned Aerial Vehicles.* Master's thesis, Air Force Instituite of Technology, 2004.

51. Miller, Peter. "Swarm Theory". *National Geographic*, July, 2007.

52. Nikovski, Daniel and Illah Nourbakhsh. "Learning Probabilistic Models for Decision-Theoretic Navigation of Mobile Robots". *Proc. 17th International Conf. on Machine Learning*, 671–678. Morgan Kaufmann, San Francisco, CA, 2000. URL `citeseer.ist.psu.edu/nikovski00learning.html`.

53. Nowak, Dustin. *Self Organized Systems in the Real World and Their Computational Implementation*. Technical report, Air force Institute of Technology, 2006.

54. Nowak, Dustin and Gary B. Lamont. "Self Organized Genetic Algorithms Applied to Self Organized Unmanned Aerial Vehicle Swarms", Dec 2007. Submitted to IEEE TEC special Edition on Swarm Intelligence.

55. Nowak, Ian Price, Dustin J. and Gary B. Lamont. "Self Organized UAV Swarm Planning Optimization for Search and Destroy Using Swarmfare Simulation". M.-H. Hsieh J. Shortle J. D. Tew S. G. Henderson, B. Biller and eds. R. R. Barton (editors), *Proceedings of the 2007 Winter Simulation Conference*. 2007.

56. Parsopoulos, K.E., D.K. Tasoulis, N.G. Pavlidis, V.P. Plagianakos, and M.N. Vrahatis. "Vector Evaluated Differential Evolution for Multiobjective Optimization". *Evolutionary Computation, 2004. CEC2004. Congress on*, 204– 211. June 2004. URL `citeseer.ist.psu.edu/parsopoulos04vector.html`.

57. Pohl, Adam. *Multi-Objective UAV Mission Planning Using Evolutionary Computation*. Master's thesis, Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 2008.

58. Poli, R., W. Langdon, and O. Holland. "Extending Particle Swarm Optimization via Genetic Programming", 2005.

59. Price, Ian C. *Evolving Self Organizing Behavior for Homogeneous and Heterogeneous Swarms of UAVs and UCAVs*. Master's thesis, Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 2006.

60. Rasmussen, Steven J. and Phillip R. Chandler. "MultiUAV: A Multiple UAV Simulation For Investigation ofCooperative Control". *Proceedings of the 2002 Winter Simulation Conference*, 2002. URL `Steven.Rasmussen@wpafb.af.mil`. San Diego, CA.

61. Raytheon. "MVCS (Multi Vehicle Control System).", 2006. URL `http://www.raytheon.com/`.

62. Reynolds, Craig W. "Flocks, herds, and schools: A distributed behavioral model". *Computer Graphics*, 21:265–280, 1987.

63. Reynolds, Craig W. "Steering Behaviors For Autonomous Characters". *Proceedings of the 2005 Winter Simulation Conference. San Jose, California*, 763–782. 2005.

64. Rosenblatt, J. K. "DAMN: A Distributed Architecture for Mobile Navigation". *Proc. of the AAAI Spring Symp. on Lessons Learned from Implemented Software Architectures for Physical Agents*. Stanford, CA, 1997. URL `citeseer.ist.psu.edu/article/rosenblatt97damn.html`.

65. Roy, N. *Finding approximate POMDP solutions through belief compression*. Ph.D. thesis, MIT, 2000. URL `citeseer.ist.psu.edu/roy00finding.html`.

66. Rubio, J.S., J. Vagners, and Rydsyk R. "Adaptive Path Planning for Autonomous UAV Oceanic Search Missions". AiAA 1st Intelligent Systems Technical Conference, 2004.

67. Russell, M.A., G.B. Lamont, and K. Melendez. "On using SPEEDES as a platform for a parallel swarm simulation". *Winter Simulation Conference, 2005 Proceedings of the*, 9pp. 4-7 Dec. 2005.

68. Russell, Stuart J. and Peter Norvig. *Artificial Intelligence: A Modern Approach 2nd Ed.* Prentice Hall, 2003.

69. Schnieder Jr., William. "Defense Science Board Study on Unmanned Aerial Vehicles and Uninhabited Combat Aerial Vehicles", 2004.

70. Secomandi, Nicola and Francois Margot. "Reoptimization Approaches for the Vehicle Routing Problem with Stochastic Demands", Apr 2007. Robert Fourer Ed. CMU.

71. Seeley, Thomas D. and P. Kirk Visscher. "Choosing a home: how the scouts in a honey bee swarm perceive the completion of their group decision making". *Behavioral Ecology and Sociobiology*, 54:511–520, 2003.

72. Sheridan, William L., Thomas B. ; Verplank. *Human and Computer Control of Undersea Teleoperators*. Technical rept., Massachusetts Institute of Technology Cambridge Man-Machine Systems Lab, JUL 1978. ADA057655.

73. Slear, J. *UAV Swarm Mission Planning and Simulation System*. Master's thesis, Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, June 2006.

74. Slear, James N. *AFIT UAV swarm mission planning and simulation system*. Master's thesis, Air Force Institute of Technology, 2006.

75. Toth, Paolo and Daniele Vigo (editors). *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2.

76. Tsujiguchi, M. and T. Odagaki. "Self-organizing social hierarchy and villages in a challenging society". *Physica A Statistical Mechanics and its Applications*, 375:317–322, February 2007.

77. Tzu, Sun. *Art Of War*. 500 B.C. URL `http://www.kimsoft.com/polwar.htm`.

78. Veldhuizen, David A. Van and Gary B. Lamont. "Multiobjective Optimization with Messy Genetic Algorithms". *Proceedings of the 2000 ACM Symposium on Applied Computing*, 470–476. ACM, Villa Olmo, Como, Italy, 2000.

79. Visscher, Thomas D. Seeley P. Kirk. "Quorum sensing during nest-site selection by honeybee swarms". *Behavioral Ecology and Sociobiology*, 56:594–601, 2004.

80. Wardell, Dean C. *Application of Fuzzy State Aggregation and Policy Hill Climbing to Multi-Agent Systems in Stochastic Thesis*. Master's thesis, Air Force Institute of Technology, 2006.

81. Wiki, Swarm Development Group. "Swarm Software". Sante Fe Institute and University of Michigan. URL `http://www.swarm.org/wiki/Main_Page`. Accessed 11 Nov 2006.

82. Wolpert, David H. and William G. Macready. "No Free Lunch Theorems for Optimization". IBM Almaden Research Center and Sante Fe Institute, 1996.

83. Woolley, Brian G. and Gilbert L. Peterson. "Genetic Evolution of Hierarchical Behavior Structures". Dirk Thierens et al. (editor), *2007 Genetic and Evolutionary Computation Conference*, volume II, 1731–1738. Associaition for Computing Machinery, 2007.

# *Index*

The index is conceptual and does not designate every occurrence of a keyword. Page numbers in bold represent concept definition or introduction.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



**PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

| **1. REPORT DATE** *(DD-MM-YYYY)* | **2. REPORT TYPE** | **3. DATES COVERED** *(From – To)* |
|---|---|---|
| 27-03-2008 | **Master's Thesis** | Aug 2006 – Mar 2008 |

**4. TITLE AND SUBTITLE**

**EXPLOITATION OF SELF ORGANIZATION IN UAV SWARMS FOR OPTIMIZATION IN COMBAT ENVIRONMENTS**

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**

Nowak, Dustin J., Capt, USAF

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)**

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENG)
2950 Hobson Way
WPAFB OH 45433-7765

**8. PERFORMING ORGANIZATION REPORT NUMBER**

AFIT/GCS/ENG/08-18

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

AFRL/SNZW
Attn: Mike Foster
2241Avionics Circle
WPAFB OH 45433-7303 DSN: 986-4899x3030

**10. SPONSOR/MONITOR'S ACRONYM(S)**

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
**APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.**

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

This investigation focuses primarily on the development of effective target engagement for unmanned aerial vehicle (UAV) swarms using autonomous self-organized cooperative control. This development required the design of a new abstract UAV swarm control model which flows from an abstract Markov structure, a Partially Observable Markov Decision Process. Self-organization features, bio-inspired attack concepts, evolutionary computation (multi-objective genetic algorithms, differential evolution), and feedback from environmental awareness are instantiated within this model. The associated decomposition technique focuses on the iterative deconstruction of the problem domain state and dynamically building-up of self organizational rules as related to the problem domain environment. Resulting emergent behaviors provide the appropriate but different dynamic activity of each UAV agent for statistically accomplishing the required multi-agent temporal attack task. The current application implementing this architecture involves both UAV flight formation behaviors and UAVs attacking targets in hostile environments. This temporal application has been quite successful in computational simulation (animation) with supporting statistical analysis. The effort reflects a considerable increase in effectiveness of UAV attacks related to a previous work with increased damage and decreased causalities. In the process of developing this capability an innovative paradigm shift in autonomous agent system design evolved. Heretofore, large dimensional agent systems were developed with an a priori fixed structure, usually with emphasis on top-down or bottom-up management, control, and sensor communication. Because of the fixed structure, extension to very large dimensional systems is generally impractical. This new autonomous self-organized approach dynamically evolves an entangled communication and cooperative control distributed architecture. This entangled architecture paradigm can be applied to the research development of various large dimensional agent based autonomous systems, military and industrial.

**15. SUBJECT TERMS**
**Unmanned Aerial Vehicles, Genetic Algorithms, Evolutionary Computation, Swarm Intelligence**

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** |
|---|---|---|---|---|---|
| **REPORT** | **ABSTRACT** | **c. THIS PAGE** | | | **Dr. Gary B. Lamont (AFIT/ENG)** |
| **U** | **U** | **U** | **UU** | 210 | **19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 x 4718; Email; gary.lamont@afit.edu |

**Standard Form 298 (Rev: 8-98)**
Prescribed by ANSI Std. Z39-18